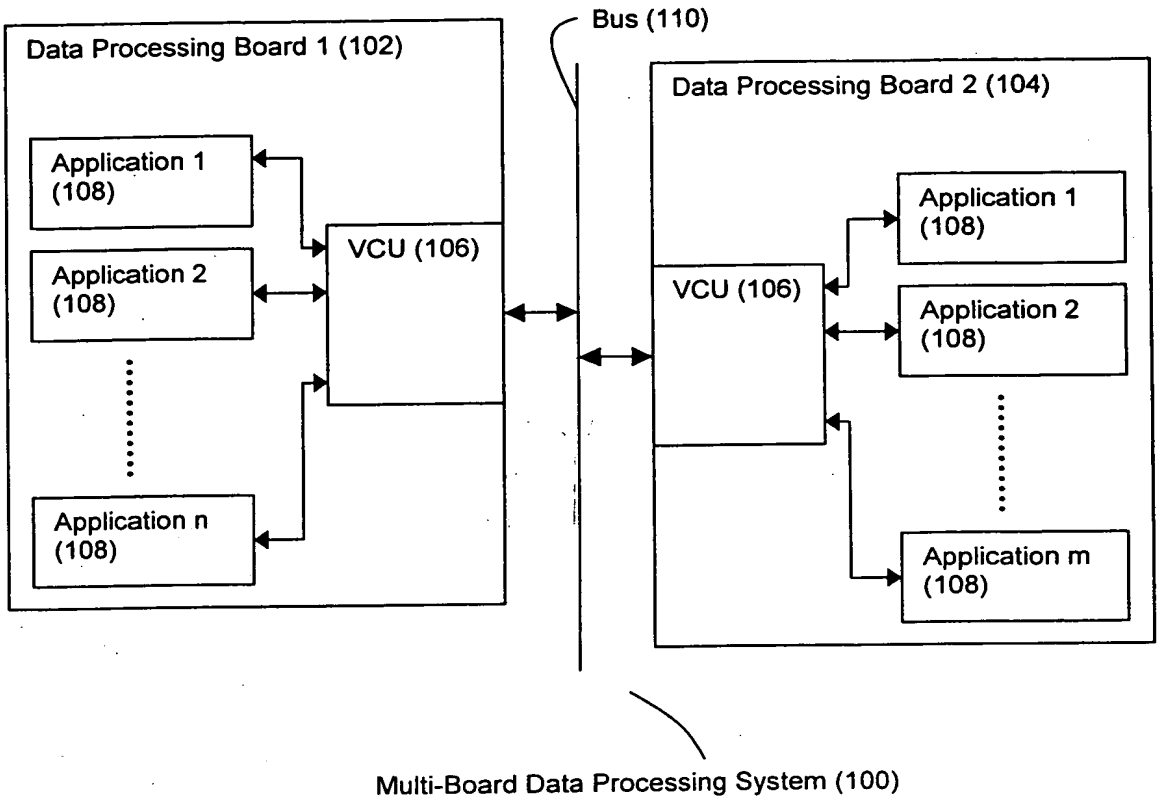

Figure 1

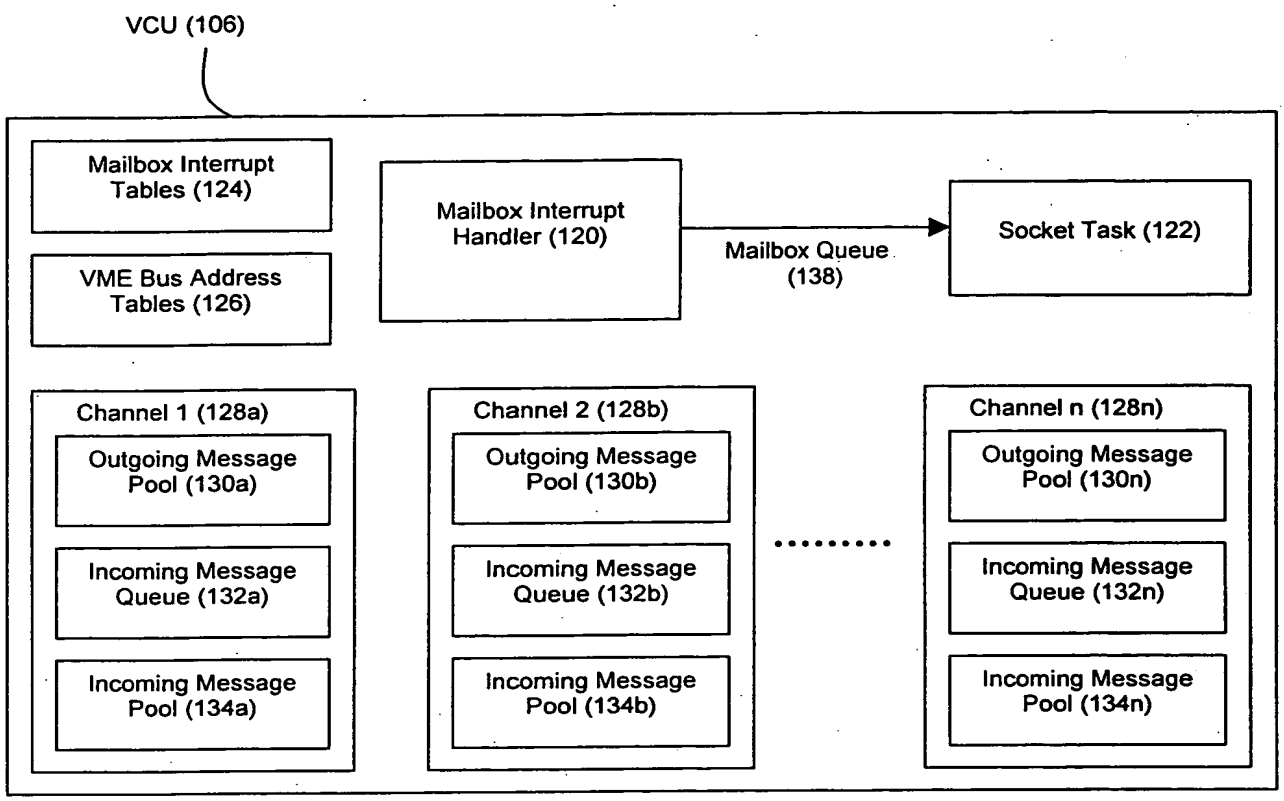
VCU (106)
Mailbox Interrupt Tables (124)
VME Bus Address Tables (126)
Mailbox Interrupt Handler (120)
Mailbox Queue (138)
Socket Task (122)
Channel 1 (128a)
Outgoing Message Pool (130a)
Incoming Message Queue (132a)
Incoming Message Pool (134a)
Channel 2 (128b)
Outgoing Message Pool (130b)
Incoming Message Queue (132b)
Incoming Message Pool (134b)
Channel n (128n)
Outgoing Message Pool (130n)
Incoming Message Queue (132n)
Incoming Message Pool (134n)

Figure 2

Return Value:	Routine Name:	Arguments:	Description:
char*	vcuAllocateBuffer(.)	int channel	Allocates a buffer with proper caching properties, sized for the specified channel
unsigned int	vcuClearErrno(.)	none	Returns the first recorded VCU error # and clears it from the error system
void	vcuCommInit(.)	none	Starts the VCU service
void	vcuCommHalt(.)	none	Forcefully halts all VCU-related tasks
int	vcuDataSize(.)	int channel	Returns the maximum message size, in bytes, for the specified channel
unsigned int	vcuErrno(.)	none	Returns the first recorded VCU error #
int	vcuFree(.)	int channel, const char *data	Marks the specified memory location in the socket Rx memory pool as usable (see detailed description)
void	vcuFreeBuffer(.)	int channel, char *buffer	Frees a buffer allocated with the vcuAllocateBuffer(.) routine
int	vcuHasPushQueueMessage(.)	int channel	Returns the number of messages in the push queue for the specified channel
int	vcuHasRxQueueMessage(.)	int channel	Returns the number of messages queued for the specified channel
int	vcuIfaceHasRxDataQueueMessage(.)	none	Returns the number of vcuRecv(.) replies queued
int	vcuIfaceHasRxRqstQueueMessage(.)	none	Returns the number of vcuRecv(.) calls queued
int	vcuIfaceHasTxQueueMessage(.)	none	Returns the number of vcuSend(.) calls queued
int	vcuIfaceHasTxReplyQueueMessage(.)	none	Returns the number of vcuSend(.) replies queued (used for push configs)
int	vcuIfaceHasTxStatusQueueMessage(.)	none	Returns the number of vcuSend(.) status messages queued
int	vcuIsMemPairPoolEmpty(.)	none	Returns 0 if the mem pair pool (used for push configs) is not empty and a non-zero value if it is empty
int	vcuIsRxPoolEmpty(.)	int channel	Returns 0 if the receive pool for the specified channel is not empty and a non-zero value if it is empty

Figure 3(a)

Return Value:	Routine Name:	Arguments:	Description:
int	vcuIsTxPoolEmpty	int channel	Returns 0 if the transmit pool for the specified channel is not empty and a non-zero value if it is empty
int	vcuPing(.)	int destination	Checks that a remote board is reachable via VCU. If more than one bit is set, a table is printed
void	vcuPrintChannelInfo(.)	int channel	Prints out the configuration information for the specified channel. If the argument is 0 (default), a table is printed
void	vcuPrintDebug(.)	int debugLevel	Triggers a printout of debugging data
int	vcuQueryForConfig(.)	int channel	Returns the configuration of the channel for internal VCU use (defined in vcuDefines.h)
unsigned int	vcuRecentErrno(.)	none	Returns the last recorded VCU error number
int	vcuRecv(.)	See Detailed Description	See Detailed Description
int	vcuRequestErrno(.)	int destination	Receives the VCU error # from a remote board. If more than one bit is set, a table is printed
int	vcuRequestTestPattern(.)	int destination, int channel, int msgSize	Requests a remote board to send a pre-defined test pattern of a requested size, and prints out success or error of delivery
int	vcuSend(.)	See Detailed Description	See Detailed Description

Figure 3(b)

Debug Level:	Description:
1	Prints out the use status of all Rx memory pools
2	Prints out the use status of all Tx memory pools
3	Prints out the use status of all Rx and Tx memory pools
4	Prints out detailed information about each slot of all Rx memory pools
5	Prints out detailed information about each slot of all Tx memory pools
6	Prints out detailed information about each slot of all Rx and Tx memory pools
7	Prints out the status of all Rx queues
8	Prints out the status of all push queues
9	Prints out detailed information about each slot of memPair memory pool
101	Prints out internal timing information if the system has been instrumented for timing

Figure 4

Configuration Type:	Sender Side:	Receiver Side:

Copy on Send	while(RUNNING){
	vcuSend(dest, chan, data, size);
	}
		while(RUNNING){
		vcuRecv(chan, pAddr, size);
		//Handle message at pAddr
		//(size in bytes length)
		vcuFree(chan, pAddr);
		}

Copy to Pool on Receive	while(RUNNING){
	vcuSend(dest, chan, data, size);
	}
		while(RUNNING){
		vcuRecv(chan, pAddr, size);
		//Handle message at pAddr
		//(size in bytes length)
		vcuFree(chan, pAddr);
		}

Copy to Buffer on Receive	while(RUNNING){
	vcuSend(dest, chan, data, size);
	}
		buf = vcuAllocateBuffer(chan)
		while(RUNNING){
		vcuRecv(chan, buf, size);
		vcuFree(chan, buf);
		}
		vcuFreeBuffer(chan, buf);
		}

Push to Pool on Receive	int pAddr = 0x0
	int id = UNIQUE_VALUE
	while(RUNNING){
	vcuSend(dest, chan, data, size, pAddr, id);
	}
		while(RUNNING){
		vcuRecv(chan, pAddr, size);
		//Handle message at pAddr
		//(size in bytes length)
		vcuFree(chan, pAddr);
		}

Push to Buffer on Receive	int pAddr = 0x0
	int id = UNIQUE_VALUE
	while(RUNNING){
	vcuSend(dest, chan, data, size, pAddr, id);
	//Changes in pAddr indicated when data
	//was read
	}
		buf = vcuAllocateBuffer(chan)
		while(RUNNING){
		vcuRecv(chan, buf, size);
		vcuFree(chan, buf);
		}
		vcuFreeBuffer(chan, buf);

Queue on Send	while(RUNNING){
	vcuSend(dest, chan, data, size);
	}
		buf = vcuAllocateBuffer(chan)
		while(RUNNING){
		vcuRecv(chan, buf, size);
		vcuFree(chan, buf);
		}
		vcuFreeBuffer(chan, buf);

Copy to Self	while(RUNNING){
	vcuSend(dest, chan, data, size);
	}
		while(RUNNING){
		vcuRecv(chan, pAddr, size);
		//Handle message at pAddr
		//(size in bytes length)
		vcuFree(chan, pAddr);
		}

Overwrite on Send	while(RUNNING){
	vcuSend(dest, chan, data, size);
	}
		while(RUNNING){
		vcuRecv(chan, pAddr, size);
		//Handle message at pAddr
		//(size in bytes length)
		}

Figure 5

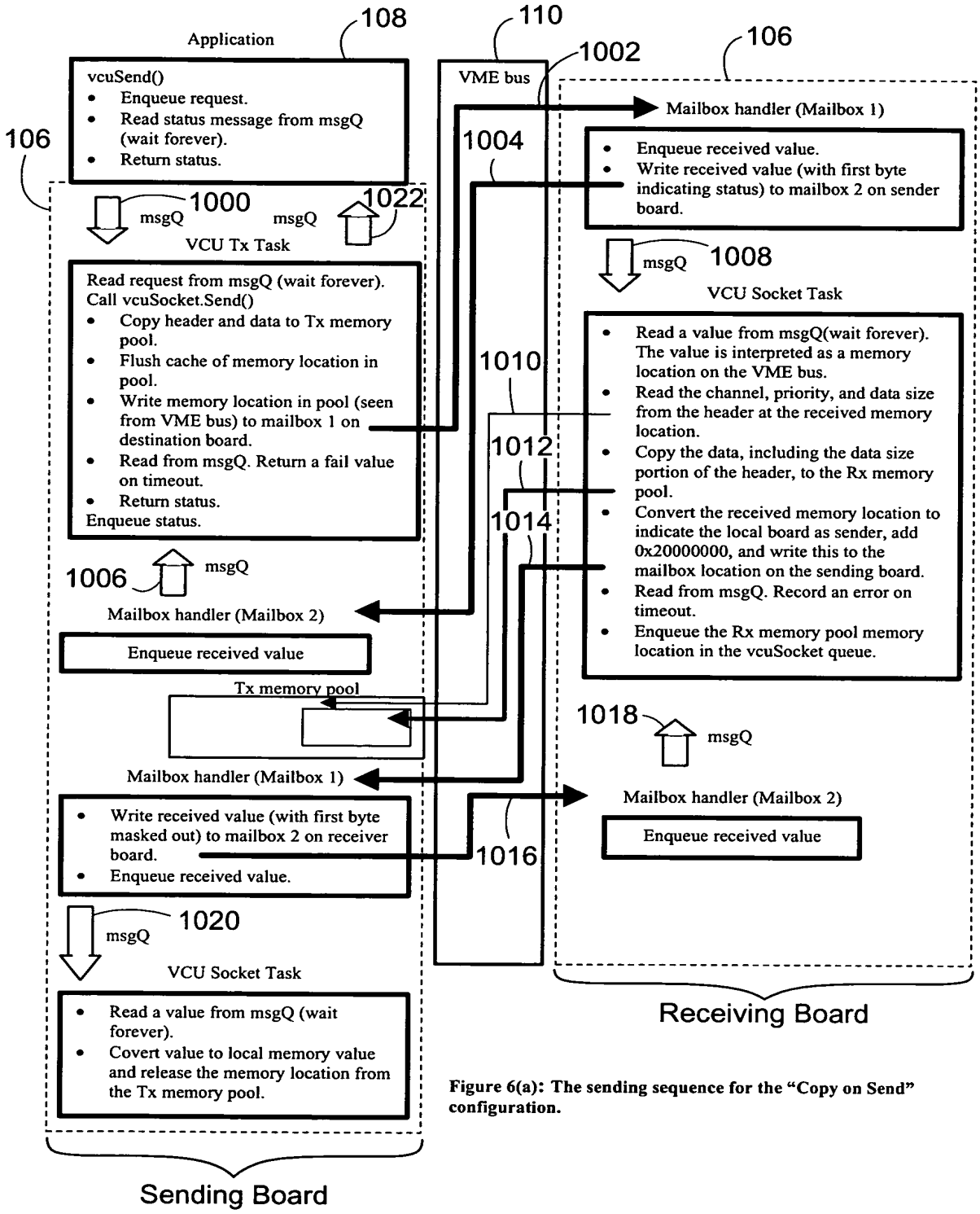

Figure 6(a): The sending sequence for the "Copy on Send" configuration.

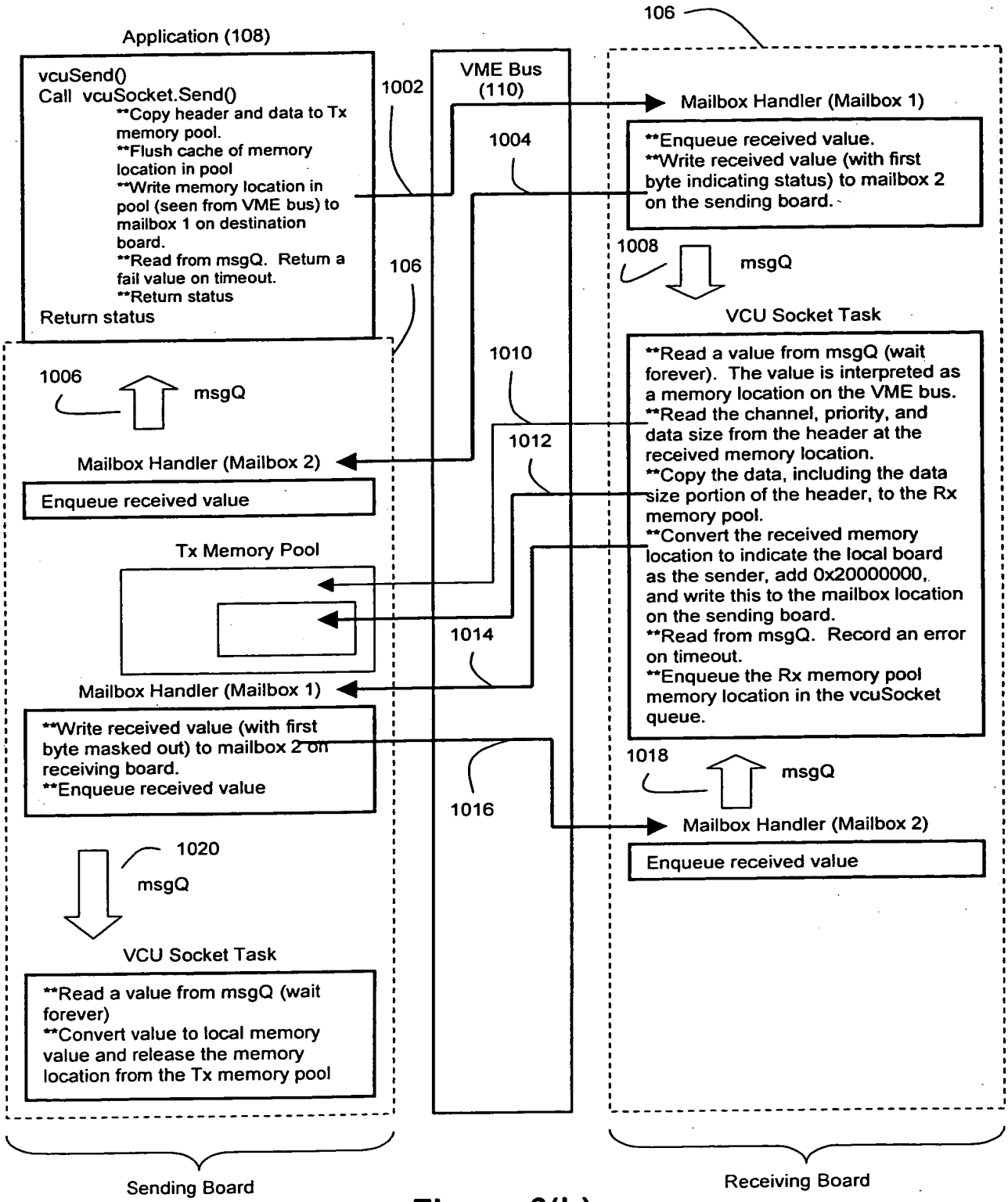

Figure 6(b)

(sending s qu nc for copy on send configuration with VCU Tx Task removed)

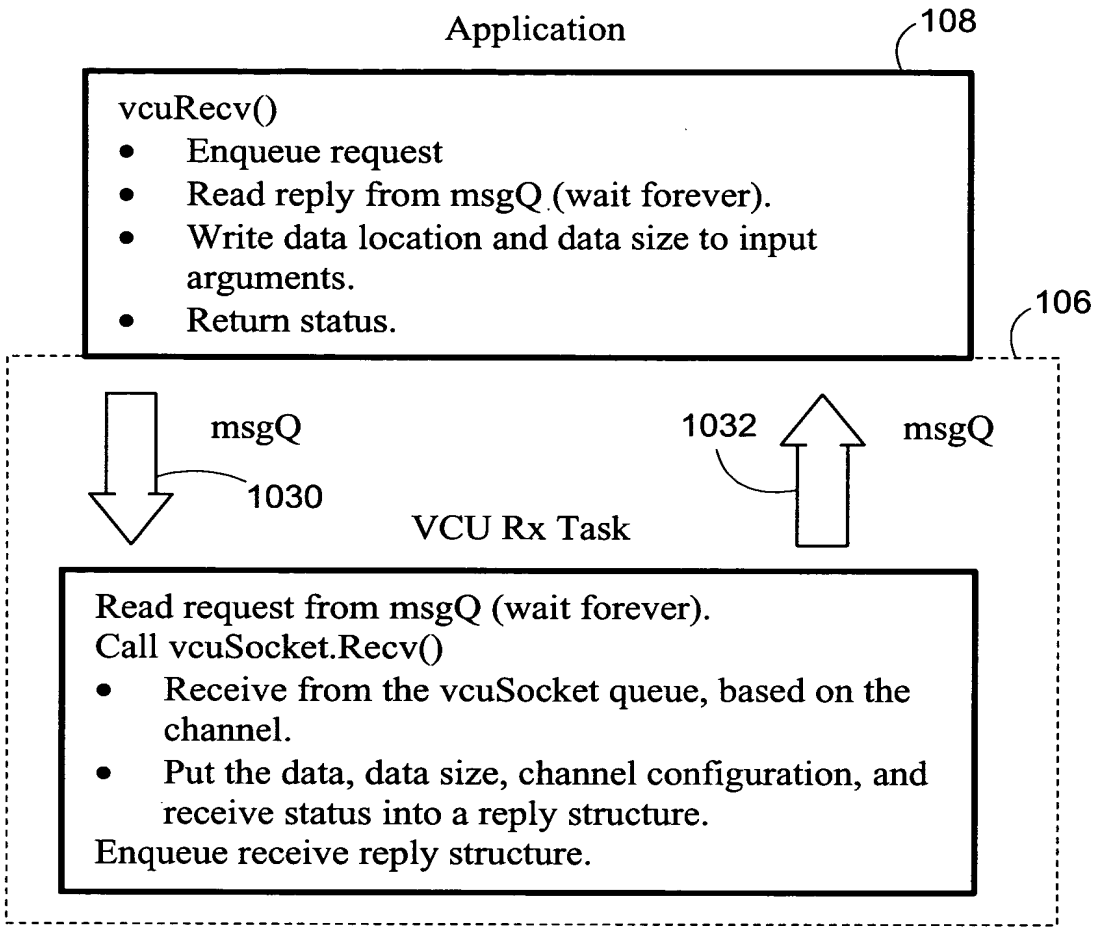

Figure 7(a): The receiving sequence for the "Copy on Send" configuration.

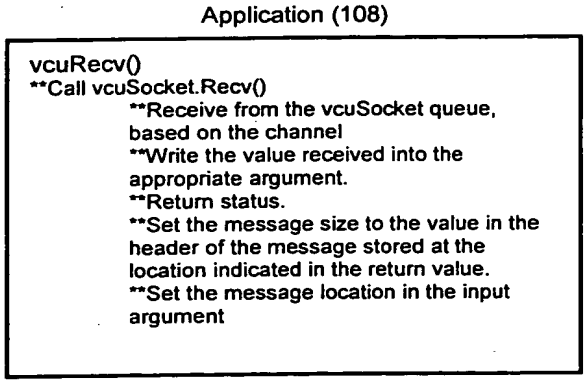

Figure 7(b)

(receiving sequence for the copy on send configuration with the VCU Rx Task removed)

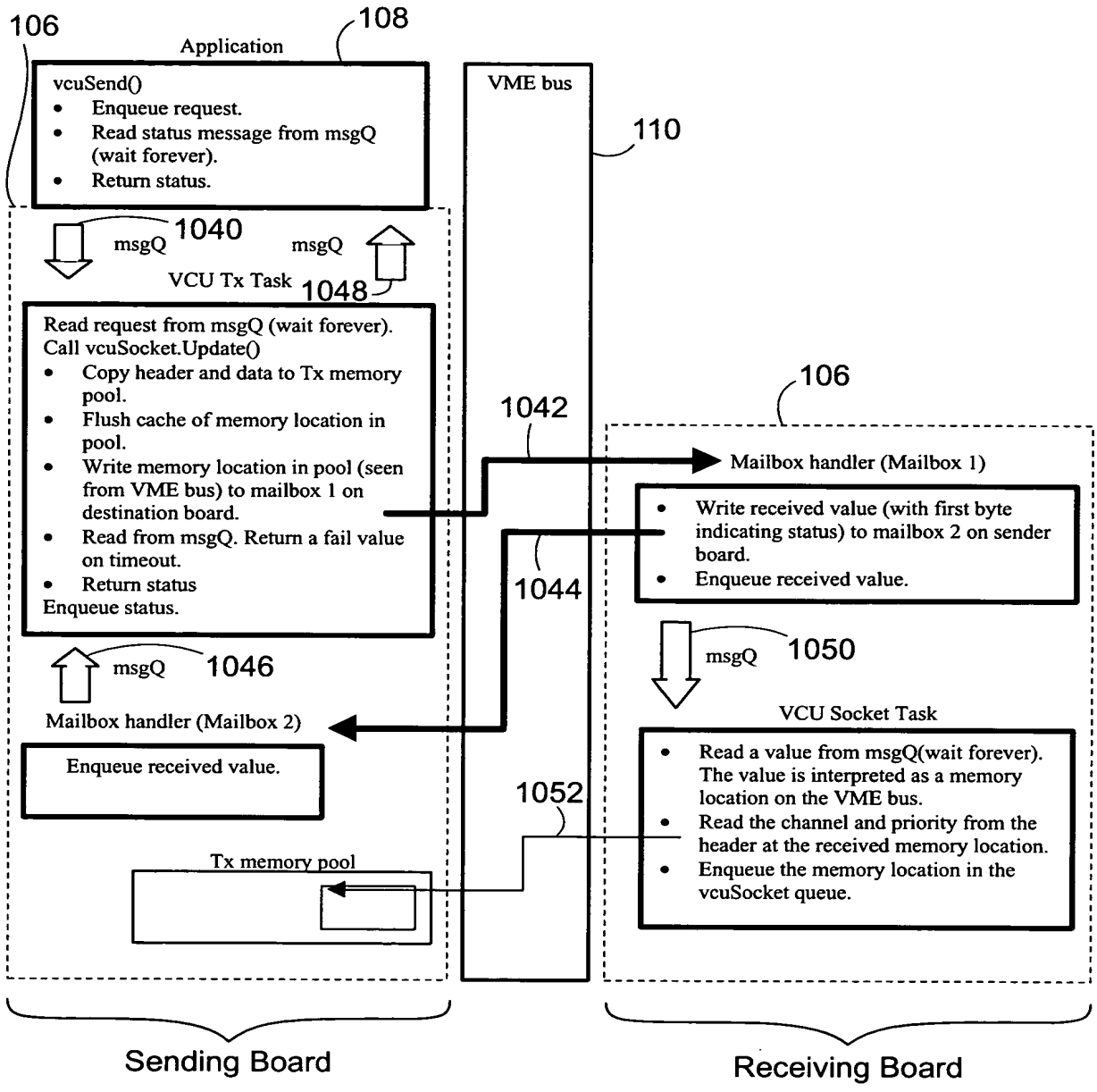

Figure 8(a): The sending sequence for the "Copy to Pool on Recv" configuration.

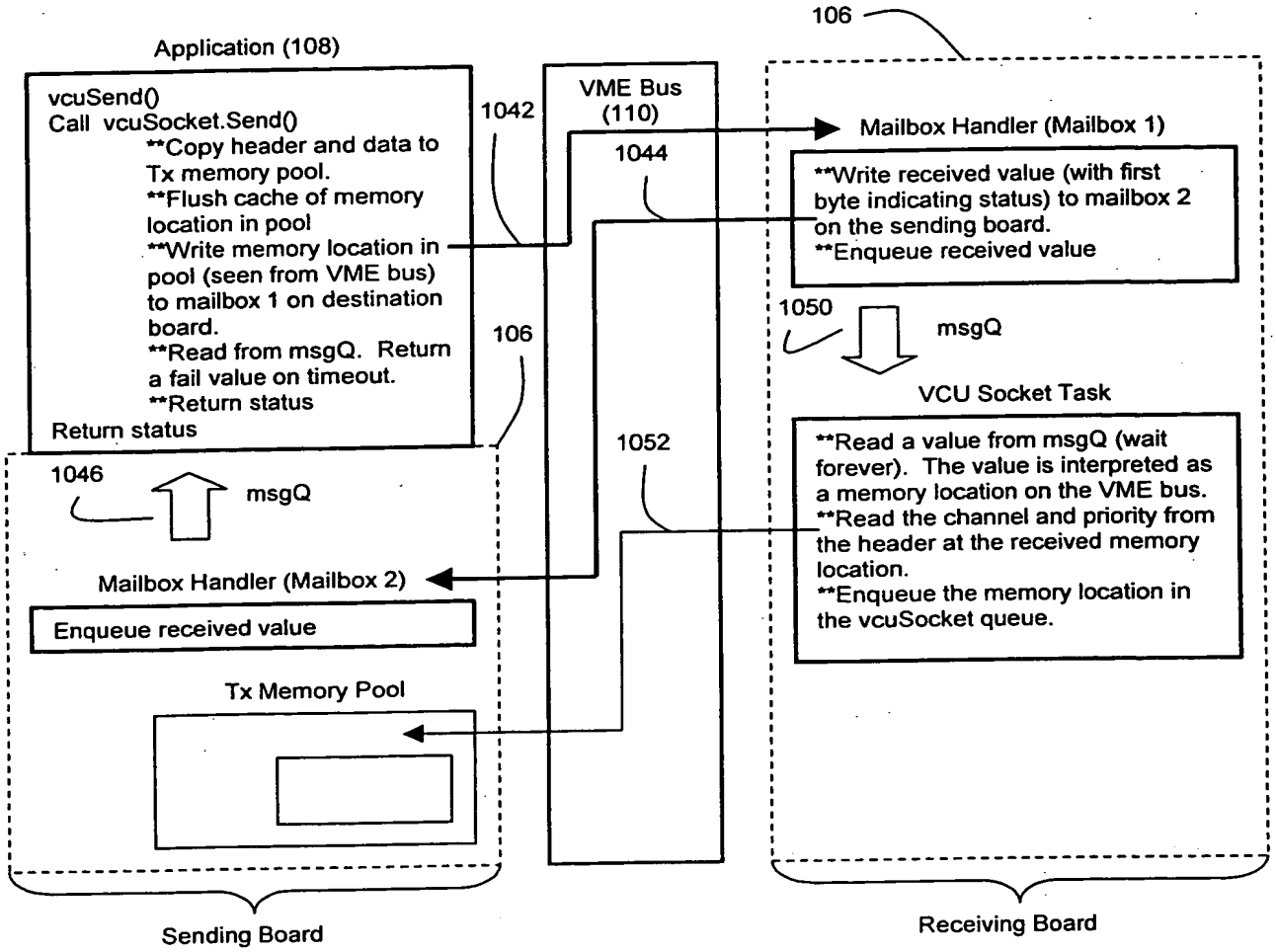

Figure 8(b)

(sending sequence for copy to pool on receive configuration)

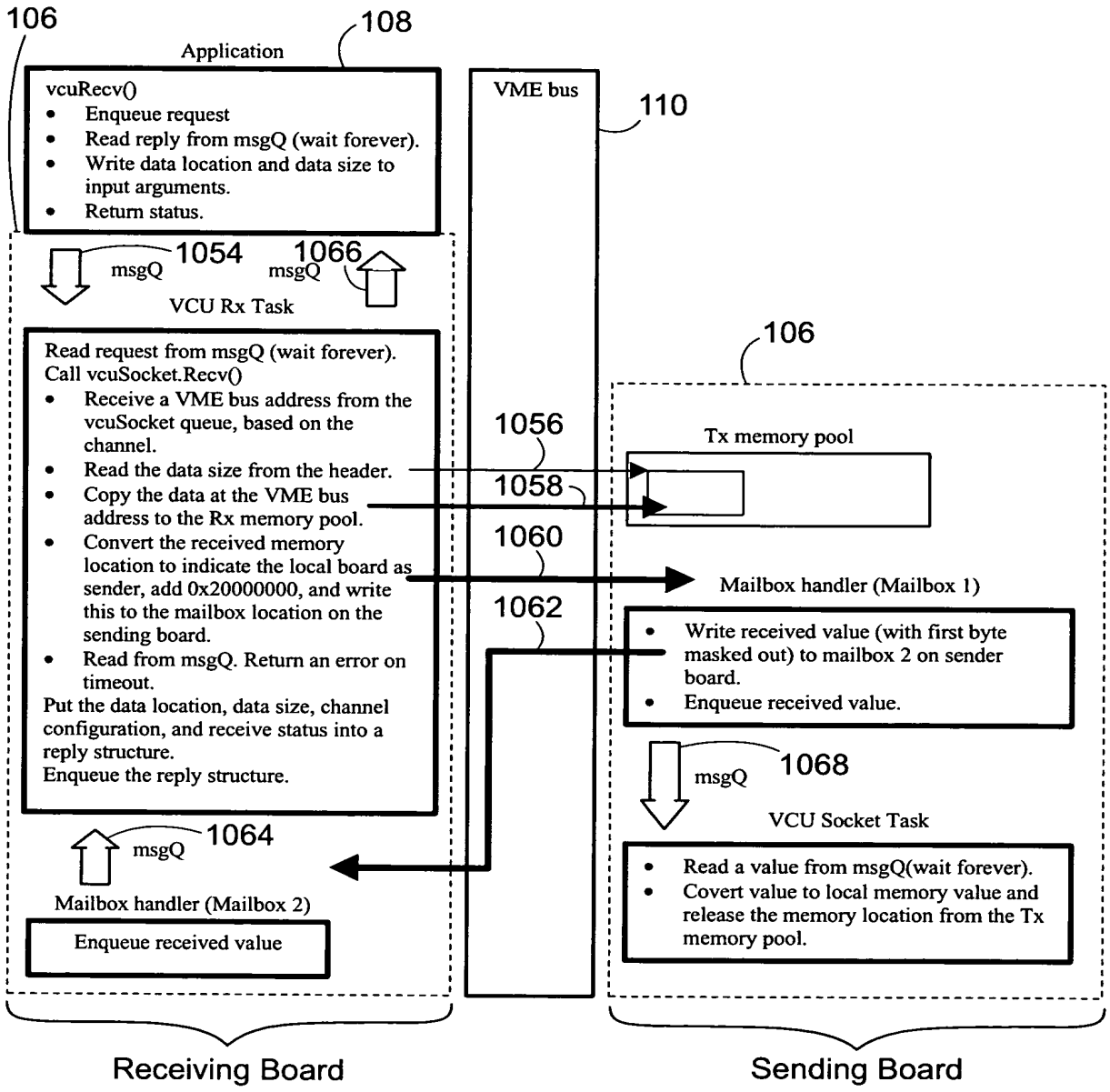

Figure 9(a): The receiving sequence for the "Copy to Pool on Recv" configuration.

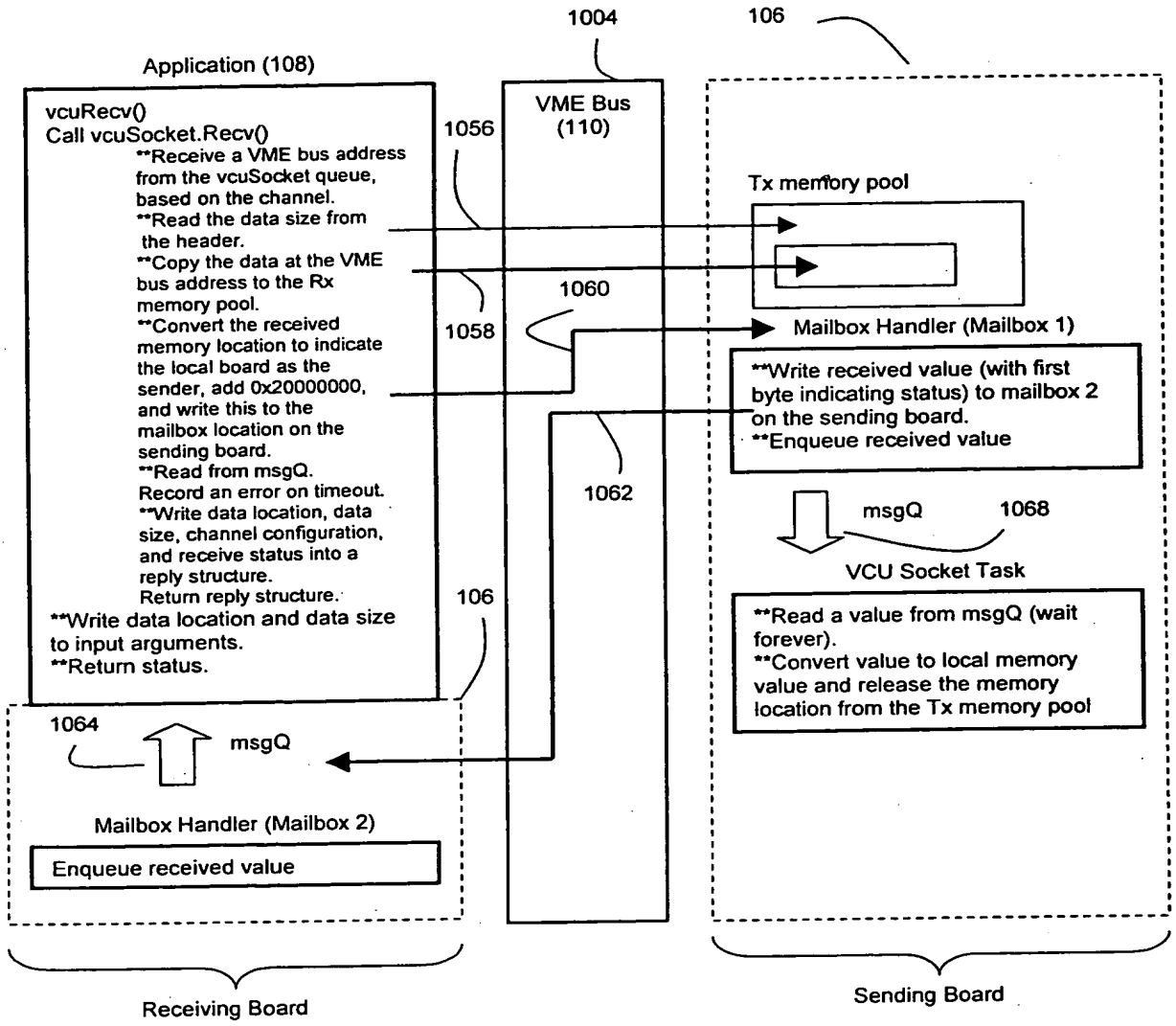

Figure 9(b)

(receiving sequence for copy to pool on receive configuration)

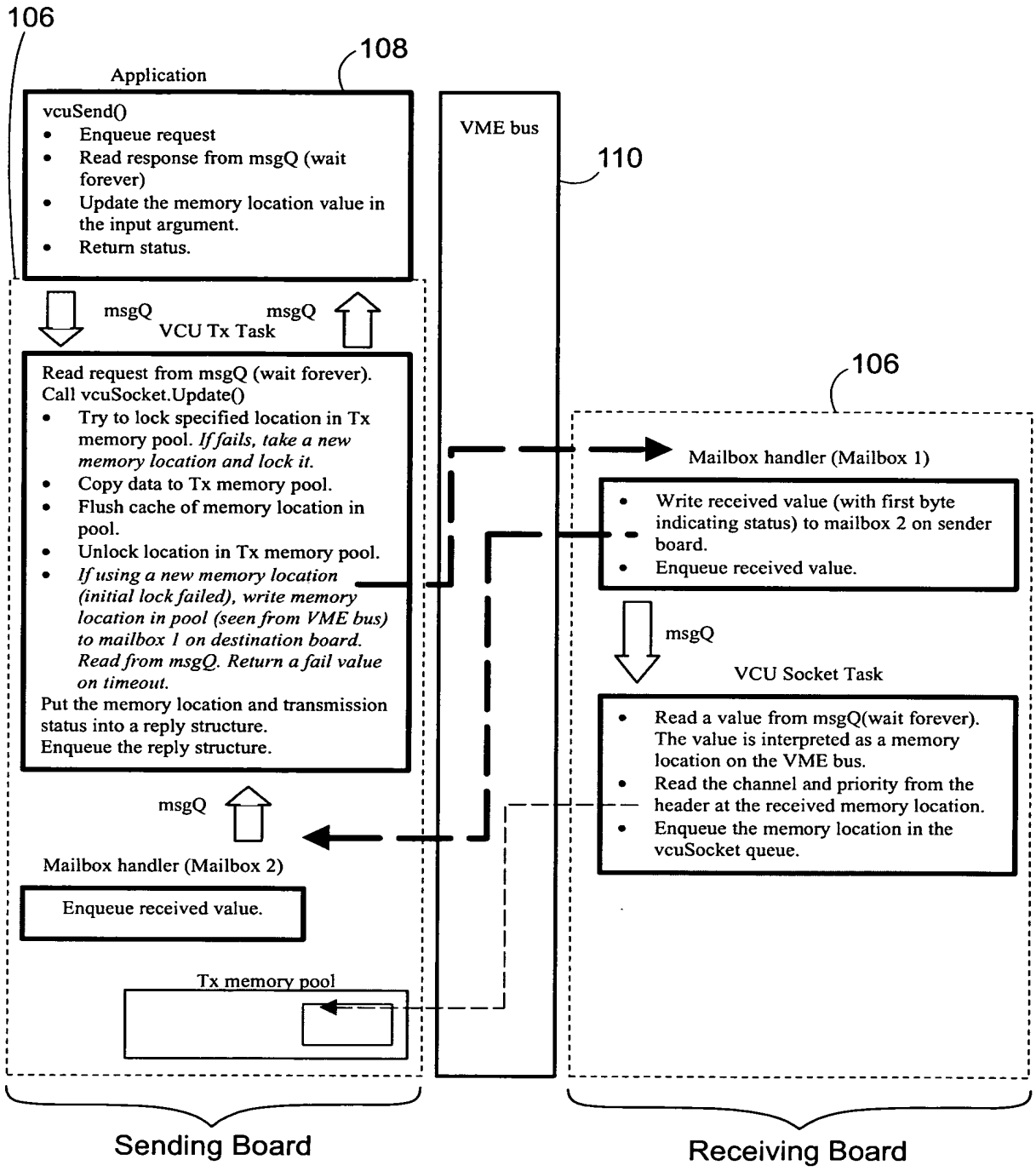

Figure 10: The sending sequence for the "Push to Pool on Recv" configuration

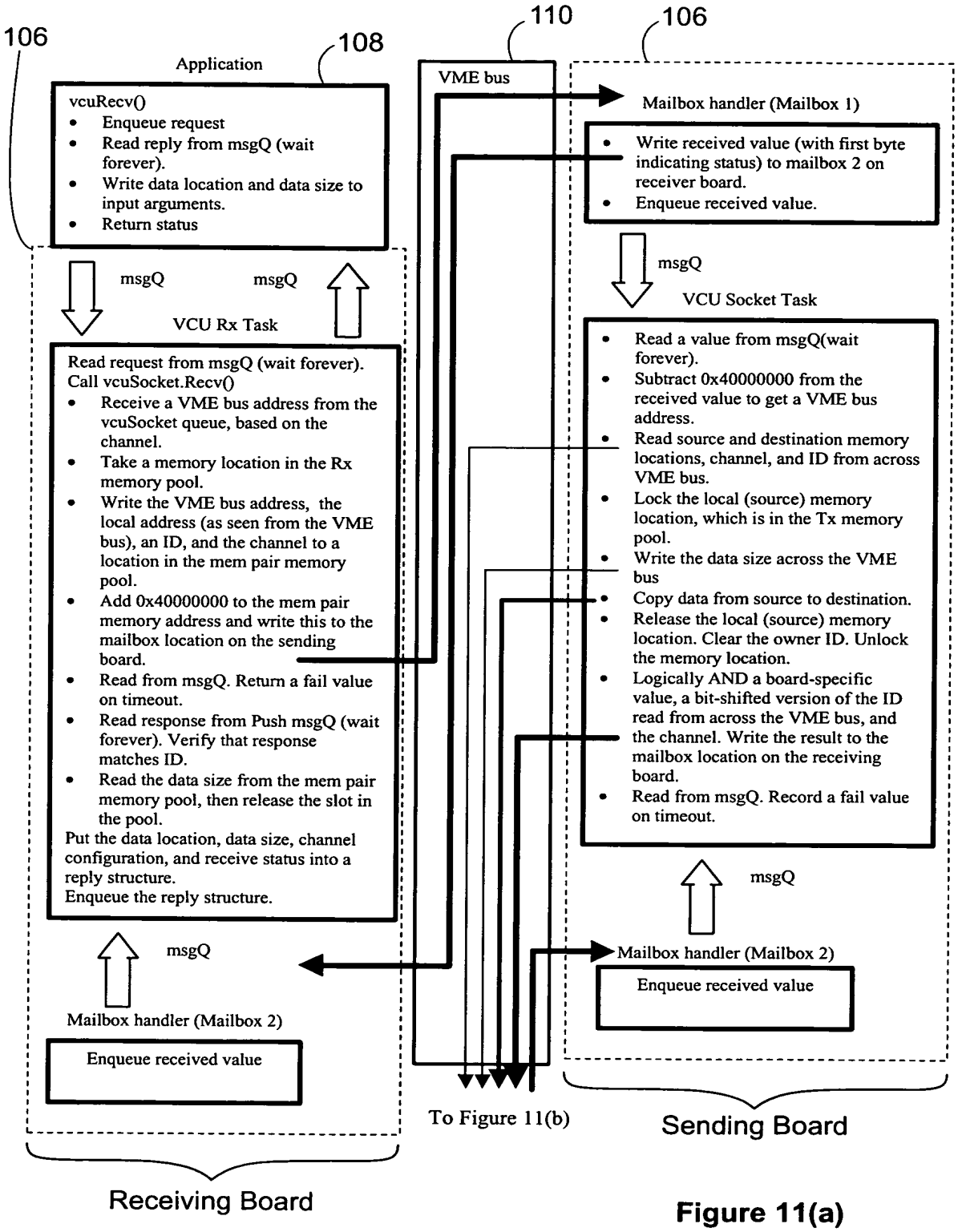
106
108
Application
vcuRecv()
• Enqueue request
• Read reply from msgQ (wait forever).
• Write data location and data size to input arguments.
• Return status
msgQ
msgQ
VCU Rx Task
Read request from msgQ (wait forever).
Call vcuSocket.Recv()
• Receive a VME bus address from the vcuSocket queue, based on the channel.
• Take a memory location in the Rx memory pool.
• Write the VME bus address, the local address (as seen from the VME bus), an ID, and the channel to a location in the mem pair memory pool.
• Add 0x40000000 to the mem pair memory address and write this to the mailbox location on the sending board.
• Read from msgQ. Return a fail value on timeout.
• Read response from Push msgQ (wait forever). Verify that response matches ID.
• Read the data size from the mem pair memory pool, then release the slot in the pool.
Put the data location, data size, channel configuration, and receive status into a reply structure.
Enqueue the reply structure.
msgQ
Mailbox handler (Mailbox 2)
Enqueue received value
Receiving Board
110
VME bus
To Figure 11(b)
106
Mailbox handler (Mailbox 1)
• Write received value (with first byte indicating status) to mailbox 2 on receiver board.
• Enqueue received value.
msgQ
VCU Socket Task
• Read a value from msgQ(wait forever).
• Subtract 0x40000000 from the received value to get a VME bus address.
• Read source and destination memory locations, channel, and ID from across VME bus.
• Lock the local (source) memory location, which is in the Tx memory pool.
• Write the data size across the VME bus
• Copy data from source to destination.
• Release the local (source) memory location. Clear the owner ID. Unlock the memory location.
• Logically AND a board-specific value, a bit-shifted version of the ID read from across the VME bus, and the channel. Write the result to the mailbox location on the receiving board.
• Read from msgQ. Record a fail value on timeout.
msgQ
Mailbox handler (Mailbox 2)
Enqueue received value
Sending Board

Figure 11(a)

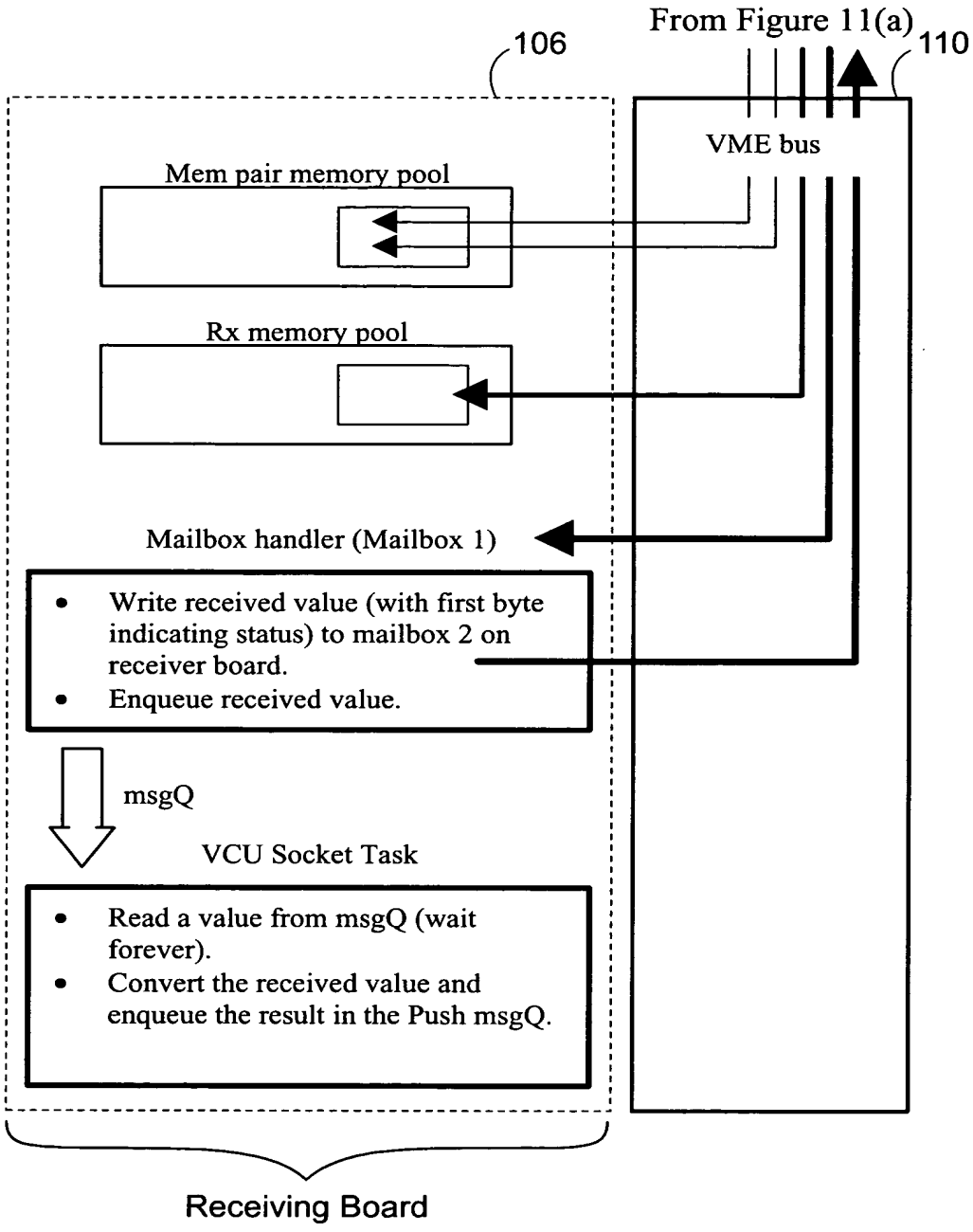

Figure 11(b): The receiving sequence for the "Push to Pool on Recv" configuration.

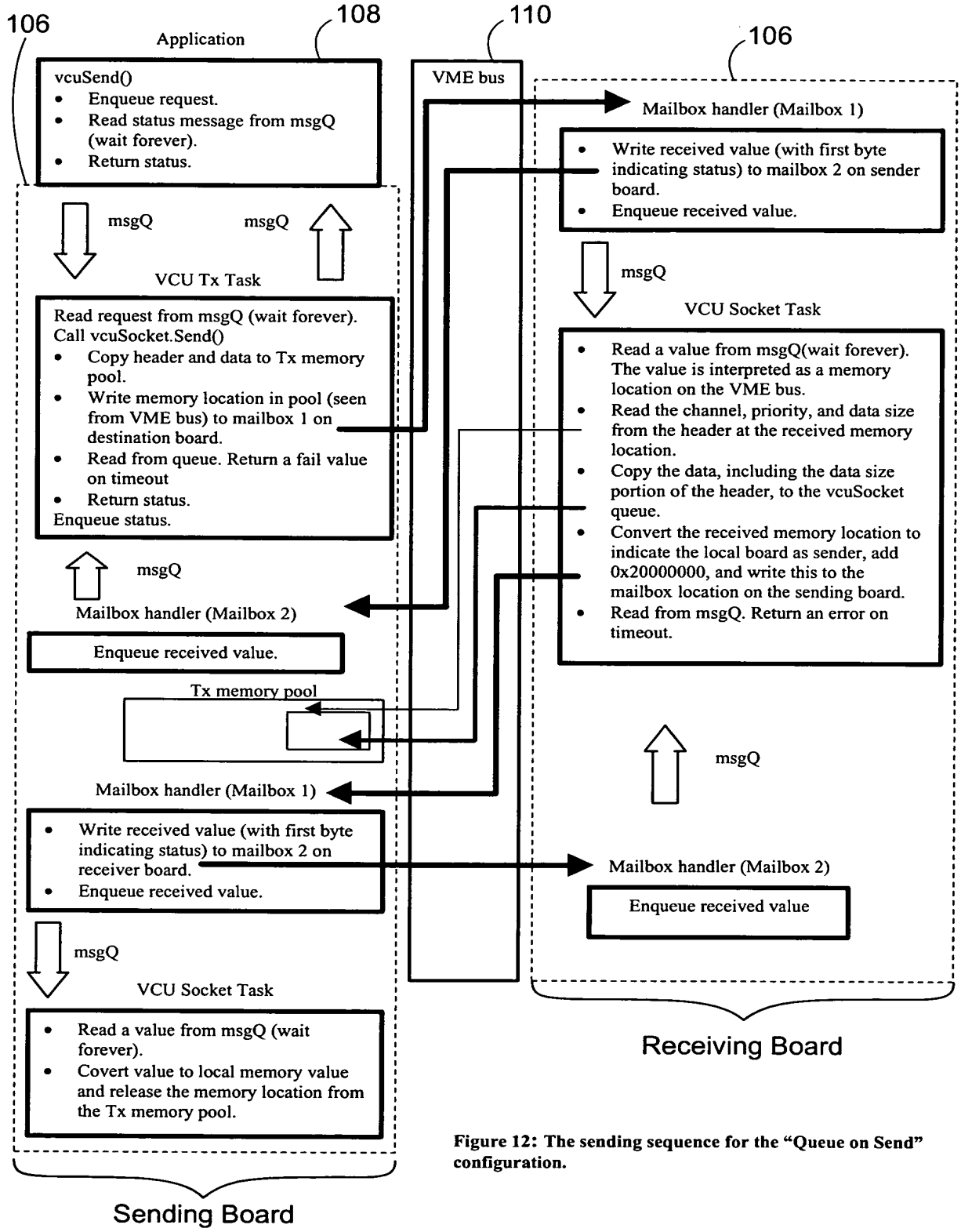

Figure 12: The sending sequence for the "Queue on Send" configuration.

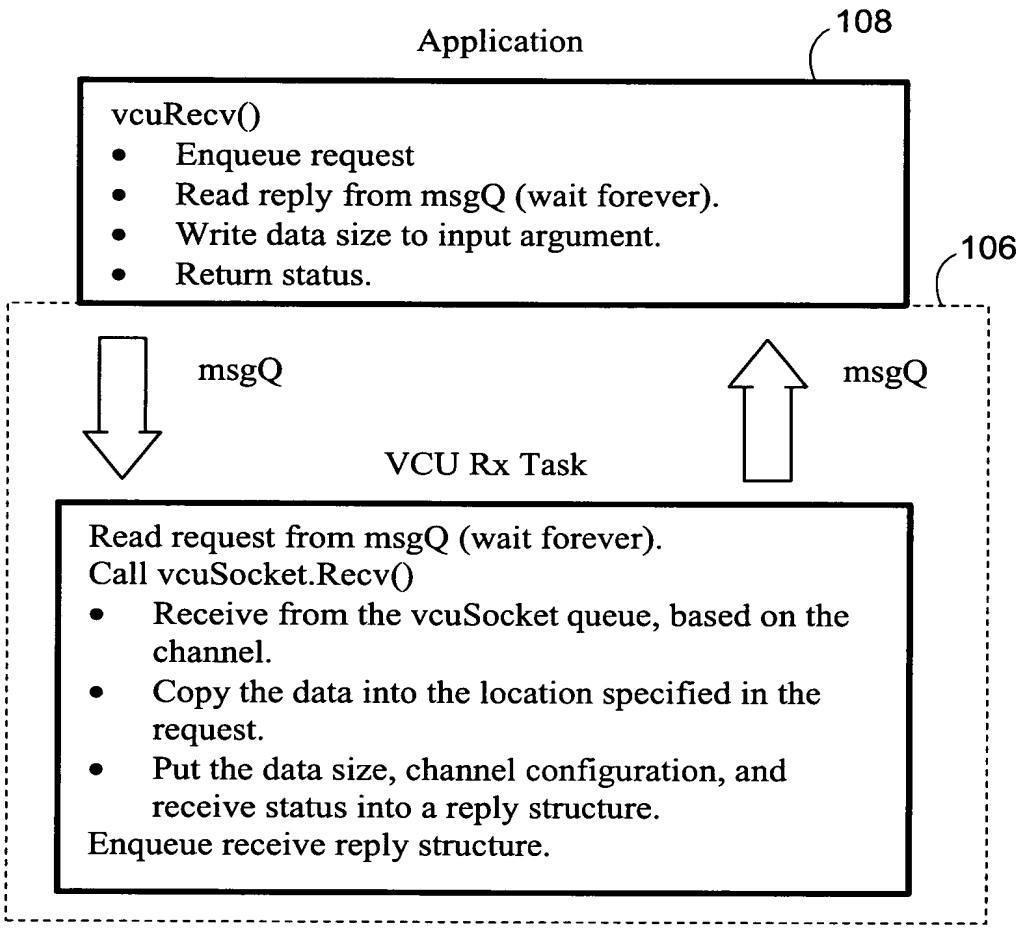

Figure 13: The receiving sequence for the "Queue on Send" configuration.

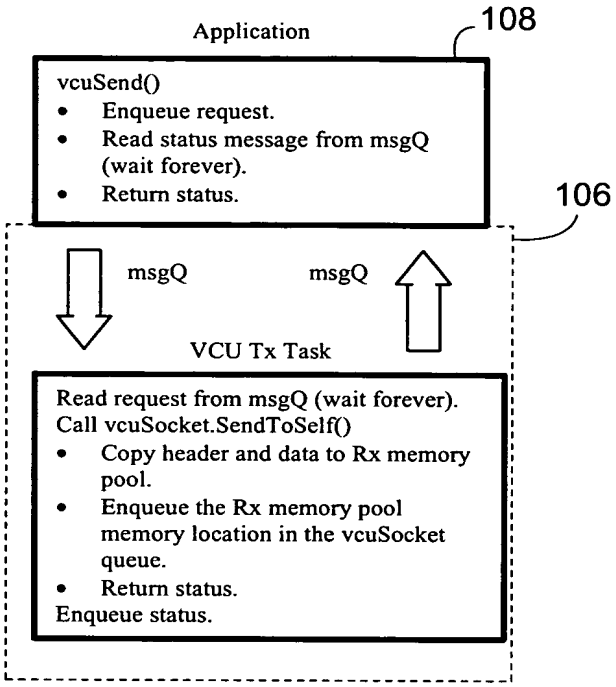

Figure 14: The sending sequence for the "Copy to Self" configuration.

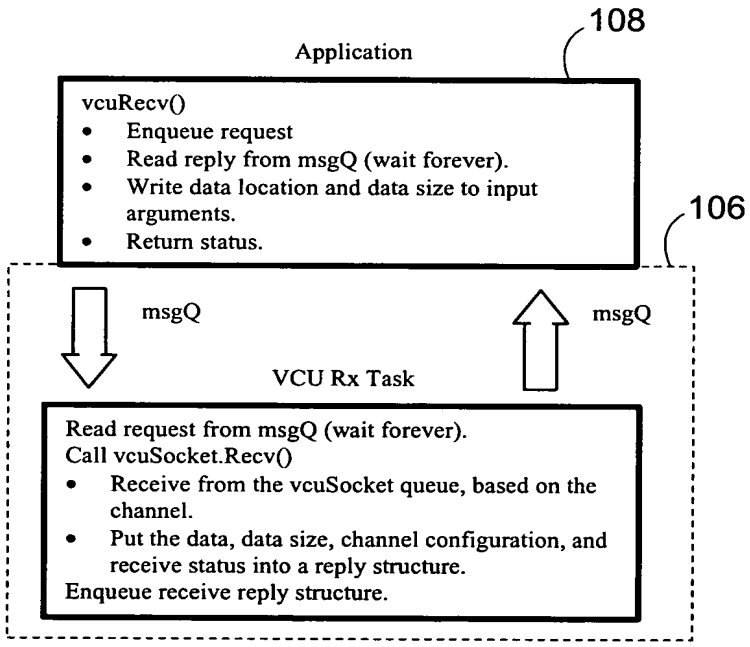

Figure 15: The receiving sequence for the "Copy to Self" configuration.

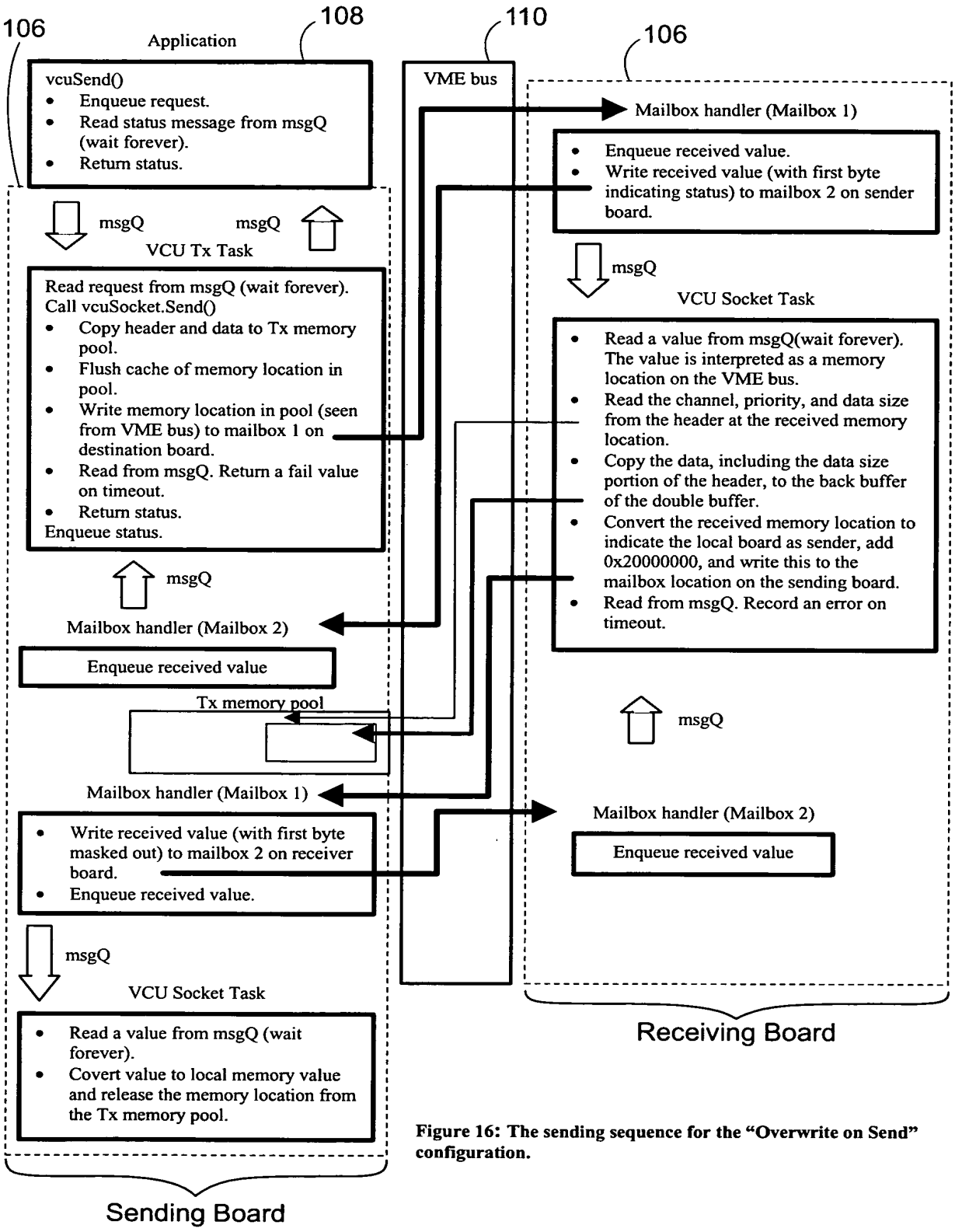

Figure 16: The sending sequence for the "Overwrite on Send" configuration.

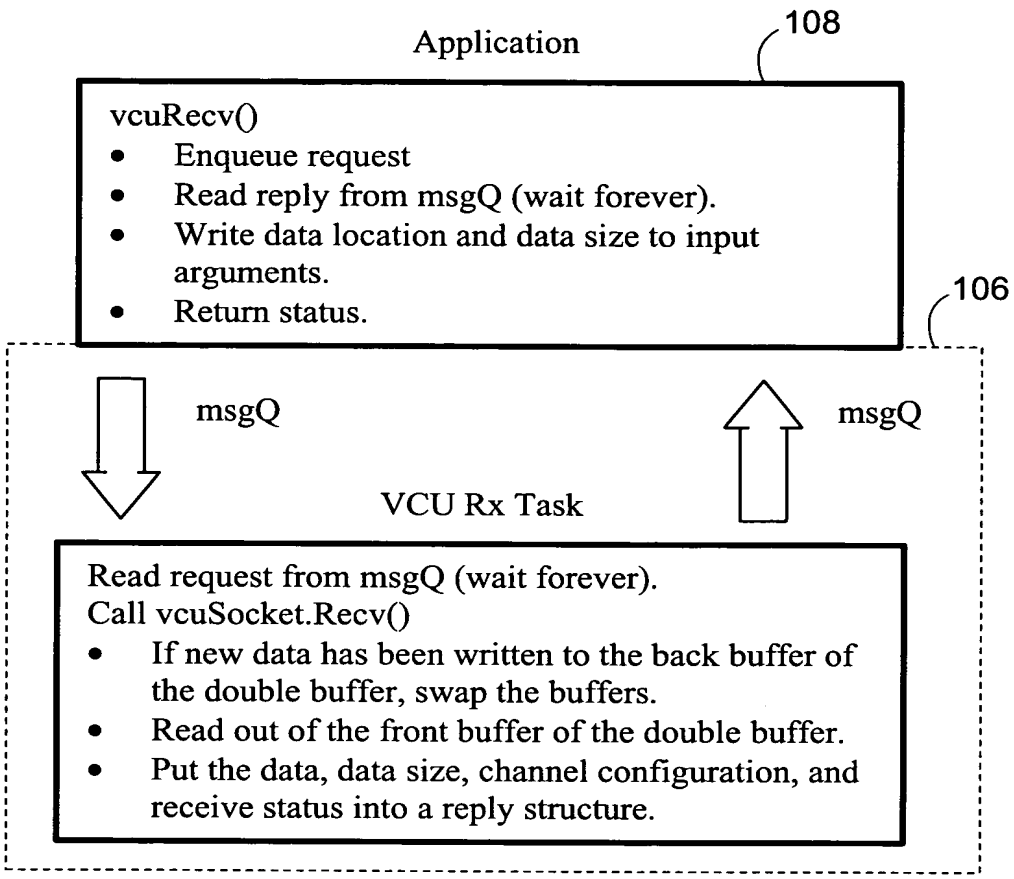

Figure 17: The receiving sequence for the "Overwrite on Send" configuration.

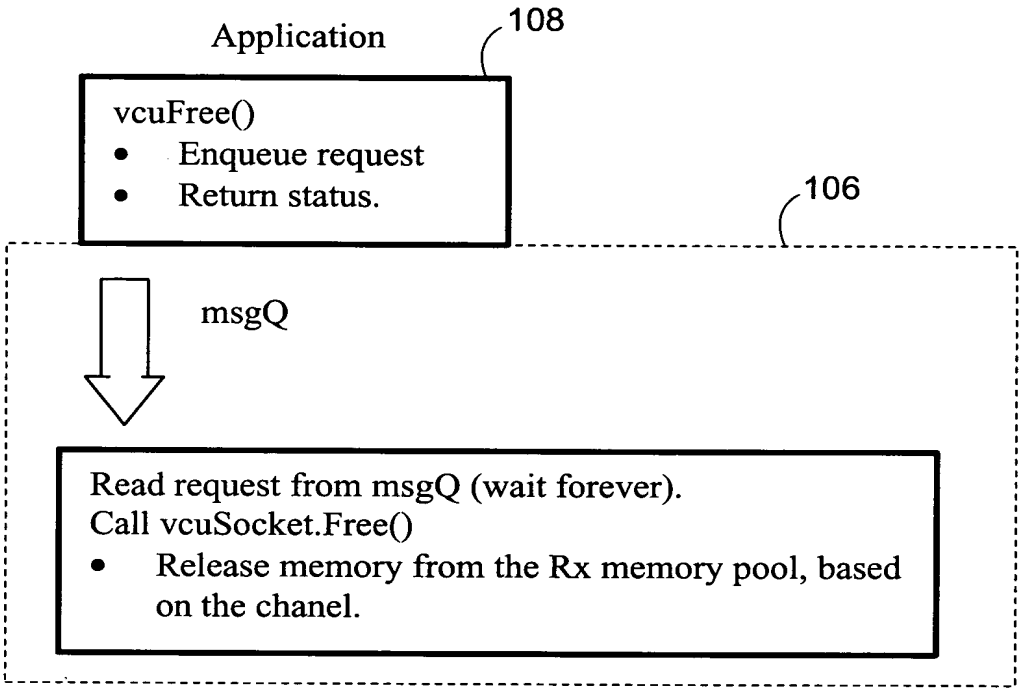

Figure 18: The freeing sequence for the receiver-side memory pool.

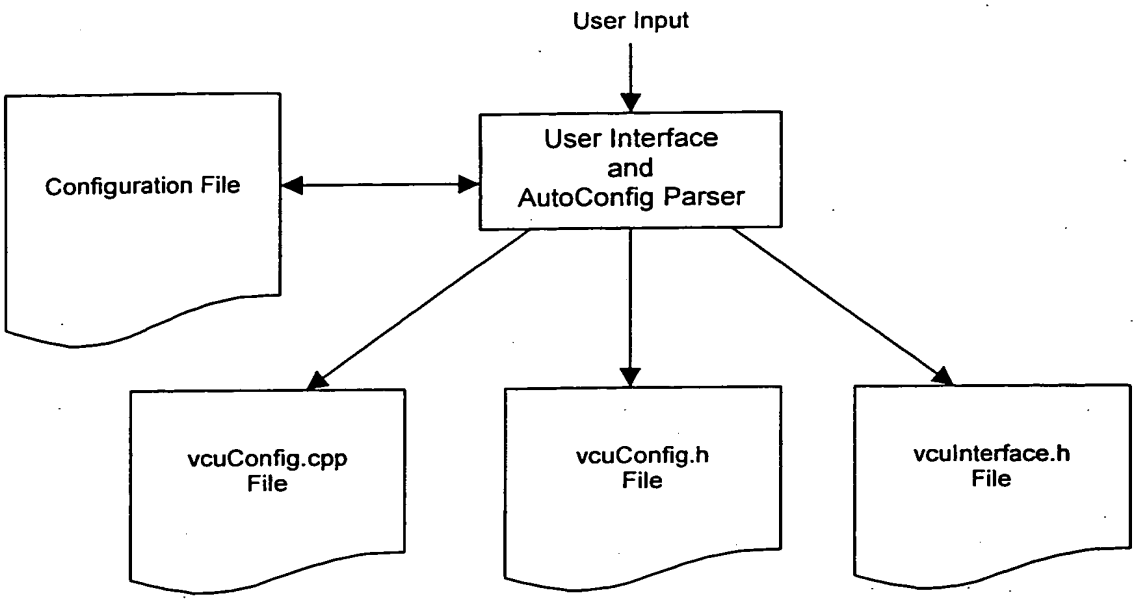

Figure 19(a)

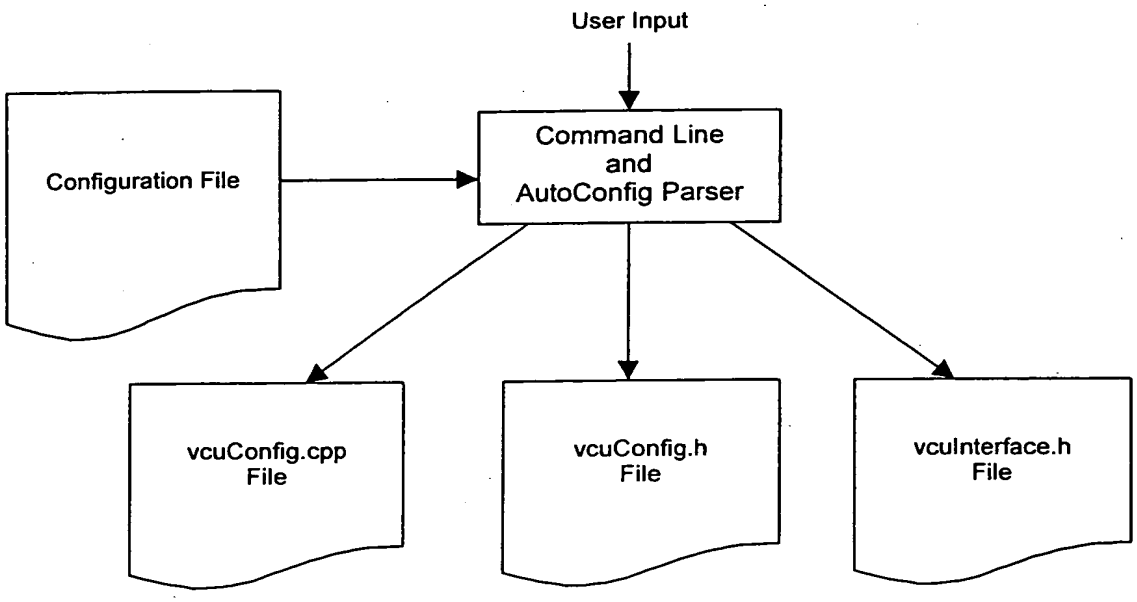

Figure 19(b)

VCU Configuration File Grammar: Backus-Naur Form (BNF)	
Top Level:	
<config_file>::=,channel_section> [<include_section>] [<handler_section>] [misc_section>]	
Description:	The configuration file should contain a channel declaration section and may optionally contain entries for include files, handler files, and other miscellaneous parameters.
Channel Declaration Section:	
<channel_section>::=Channels *number* <channel_statements>	
Description:	The channel declaration should begin with the keyword "Channels" followed by a number representing the number of channels in the system. This is then followed by said number of channel declaration statements.
Channel Declaration Statements:	
<channel_statements>::=<channel_statement> | <channel_statement> <channel_statements>	
<channel_statement>::=Chan *number* <channel_data_type> *channel_name* <channel_type>	
<channel_data_type>::=[struct | class] *custom_type* | untyped *byte_count*	
<channel_type>::=recvbuf | send | pushbuf |recvpool | pushpool |queue | self] overwrite | qself	
Description:	A channel statement provides the channel *number* followed by the channel data type which may be a structure or class of user-defined *custom_type*, or an untyped string of *byte_count* bytes. The last two tokens in a channel statement should be the user-defined *channel_name* and the channel_type. There also should be as many channel statements as the number of channels declared in the beginning of the channel declaration section.
Include File Section:	
<include_section>::= Include *number* <include_statements>	
<include_statements>::=,include_statement> | <include_statement> <include_statements>	
<include_statement>::= Include *include_file*	
Description:	The include file section provides a number of filenames that should be "included" by the file **vcuInterface.h** generated by the VCU configuration utility. The section must begin with the keyword "Include" followed by a *number* representing the number of include files in the configuration file. This is then followed by said number of include statements. Each include statement starts with the keyword "Include" and is followed by the filename *include_file* which should be "included" by **vcuInterface.h**.
Handler File Section:	
<handler_section>::=Handlers *number* <handler_statements>	
<handler_statements>::=<handler_statement> | <handler_statement> <handler_statements>	
<handler_statement>::=Hndlr *handlr_file*	
Description:	The handler file section provides a number of filenames that should be "included" by the file **vcuConfig.cpp** generated by the VCU configuration utility. The section must begin with the keyword "Handlers" followed by a *number* representing a number of handler files in the configuration file. This is then followed by said number of handler statements. Each handler statement starts with the keyword "Handlr" and is followed by the filename *handler_file* which should be "included" by the **vcuConfig.cpp** file.

Figure 20(a)

VCU Configuration File Grammar: Backus-Naur Form (BNF) (cont.)	
Miscellaneous Parameters Section:	
<misc_section>::=[<pool_parameters] [<single_parameters>] [<multiple_parameters>]	
<single parameters>::=[<ackmailbox_parameter>] [<autoackoff_parameter>] [<autoackon_parameter>] [datamailbox_parameter>] [<debugon_parameter>] [<mailboxqueuesize_parameter>] [<mempairpoolsize_parameter>] [<partitionaddr_parameter>] [<partitionfactor_parameter>] [<slavewinsize_parameter>] [<taskpriority_parameter>]	
<multiple_parameters>::=<multiple_parameter>|<multiple_parameter> <multiple_parameters>	
<multiple_parameter>::=<dma_parameter>|<dmastart_parameter>|<event_parameter>||<guaranteeddel_parameter>| <rxpoolsize_parameter>|<rxqueuesize_parameter>|<txpoolsize_parameter>	
Description:	Single parameter options should occur only once per configuration file, while multiple parameter options may occur several times, usually on a per-channel basis.
Pool Parameters:	
<pool_parameters>::=BoardCount *number* [<norxpool_parameters>] [<notxpool_parameters>]	
<norxpool_parameters>::=<norxpool_parameter>|<norxpool_parameter><norxpool_parameters>	
<notxpool_parameters>::=<notxpool_parameter>|<notxpool_parameter><notxpool_parameters>	
<norxpool_parameter>::=NoRxPool *chan_number board_processor_number*	
<notxpool_parameter>::=NoTxPool *chan_number board_processor_number*	
Description:	The pool parameters indicate which, if any, channel/board combinations should not allow receiving or transmitting respectively. The parameter list begins with the "BoardCount" keyword followed by the *number* of boards, or system processors, in the system. This is followed by one or more "NoRxPool" or "NoTxPool" keywords which tell the system which channel/board pairings, indicated by *chan_number* and *board_processor_number*, should not allow receiving and transmitting respectively.
AckMailbox Parameter:	
<ackmailbox_parameter>::=AckMailbox *number*	
Description:	This option indicates which mailbox, represented by number, should be used for automatic acknowledgements. This value must be different from the DataMailbox value, and defaults to 2.
AutoAckOff Parameter:	
<autoackoff_parameter>::=AutoAckOff	
Description:	This option indicates that auto acknowledgement should not be used in the VCU system.
AutoAckOn Parameter:	
<autoackon_parameter>::=AutoAckOn *floating_point_number*	
Description:	This option turns on the automatic acknowledgement feature of the VCU system. The floating_point_number indicates the timeout value in seconds before the message is assumed lost. This value defaults to 0.05 seconds.
DataMailbox Parameter:	
<datamailbox_parameter>::=DataMailbox *number*	
Description:	This indicates which mailbox, represented by number, should be used for inter-board communication. This value must be different from the AckMailbox value, and defaults to 1.

Figure 20(b)

VCU Configuration File Grammar: Backus-Naur Form (BNF) (cont.)
Miscellaneous Parameters Section (cont.):
Debug Parameter:
<debugon_parameter>::=DebugOn
Description: This option turns on the printing to console all values received in the data mailbox interrupt.
Mailbox Queue Size Parameter:
<mailboxqueuesize_parameter>::=MailboxQueueSize *size*
Description: This option sets the *size* of the message queue leaving the mailbox ISR. The default size is 8.
Memory Pair Pool Size Parameter:
<mempairpoolsize_parameter>::=MemPairPoolSize *size*
Description: This option sets the *size* of the memory pool for memory pairs used by all "push" channels on the board. The default size is 8.
Partition Address Parameter:
<partitionaddr_parameter>::=PartitionAddress *addr*
Description: This option sets the location of the memory partition created by the VCU for VME bus accessibility to the value specified in *addr.*
Partition Factor Parameter:
<partitionfactor_parameter>::=PartitionFactor *factor*
Description: This option sets the *factor* used in setting the size of the partition.
Slave Window Size Parameter:
<slavewinsize_parameter>::=SlaveWindowSize half | SlaveWindowSize full
Description: This option sets the size of the VME slave windows for boards in the system. "Half" sets the size to 0x04000000 and "full" to 0x08000000, with the default being half.
Task Priority Parameter:
<taskpriority_parameter>::=TaskPriority *priority*
Description: This option sets the *priority* for the VCU socket task.
DMA Parameter:
<dma_parameter>::=DMA *chan_number*
Description: This option turns forces the indicated *chan_number* to always use DMA.
DMA Start Parameter:
<dmastart_parameter>::=DMAStart *chan_number message_size*
Description: This option forces the indicated *chan_number* to always use DMA for messages that have sizes that are greater than or equal to *message_size* bytes.

Figure 20(c)

VCU Configuration File Grammar: Backus-Naur Form (BNF) (cont.)	
Miscellaneous Parameters Section (cont.):	
Event Parameter:	
<event_parameter>::=Event *chan_number event_handler*	
Description:	This option specifies the routine (*event_handler*) to be called when a board receives a message for the channel indicated by *channel_number*.
Guaranteed Delivery Parameter:	
<guaranteeddel_parameter>::=GuaranteedDelivery *chan_number*	
Description:	This option forces the indicated *chan_number* to block on certain queue and pool overflows rather than risk losing the message.
Receive Pool Size Parameter:	
<rxpoolsize_parameter>::=RxPoolSize *chan_number size*	
Description:	This option sets the *size* of the receiving memory pool for the indicated *chan_number*. The default value is 8.
Receive Queue Size Parameter:	
<rxqueuesize_parameter>::=RxQueueSize *chan_number size*	
Description:	This option sets the *size* of the receiving message queue for the indicated *chan_number*. The default value is 8.
Transmit Pool Size Parameter:	
<txpoolsize_parameter>::=TxPoolSize *chan_number size*	
Description:	This option sets the *size* of the transmitting memory pool for the indicated *chan_number*. The default value is 8.

Figure 20(d)

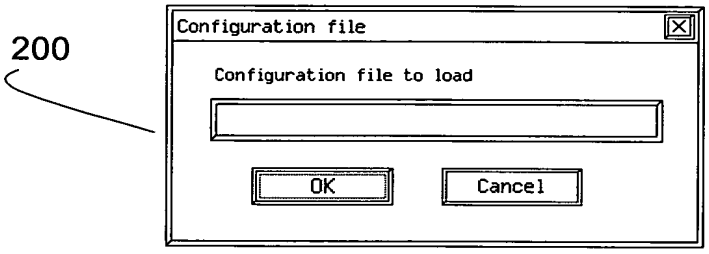

Figure 21: Initial Dialog Window for the GUI

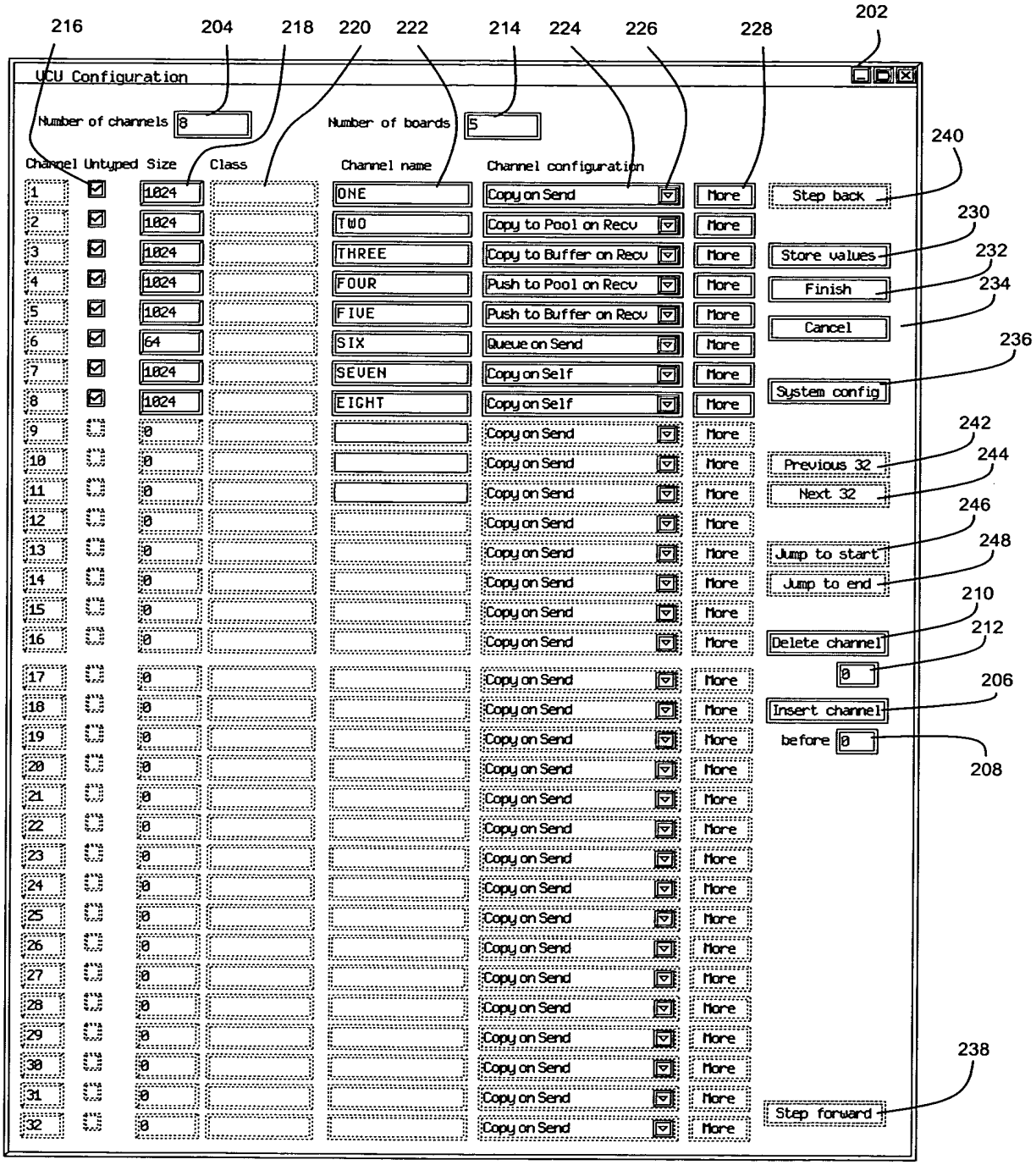

Figure 22: Primary dialog window of the GUI

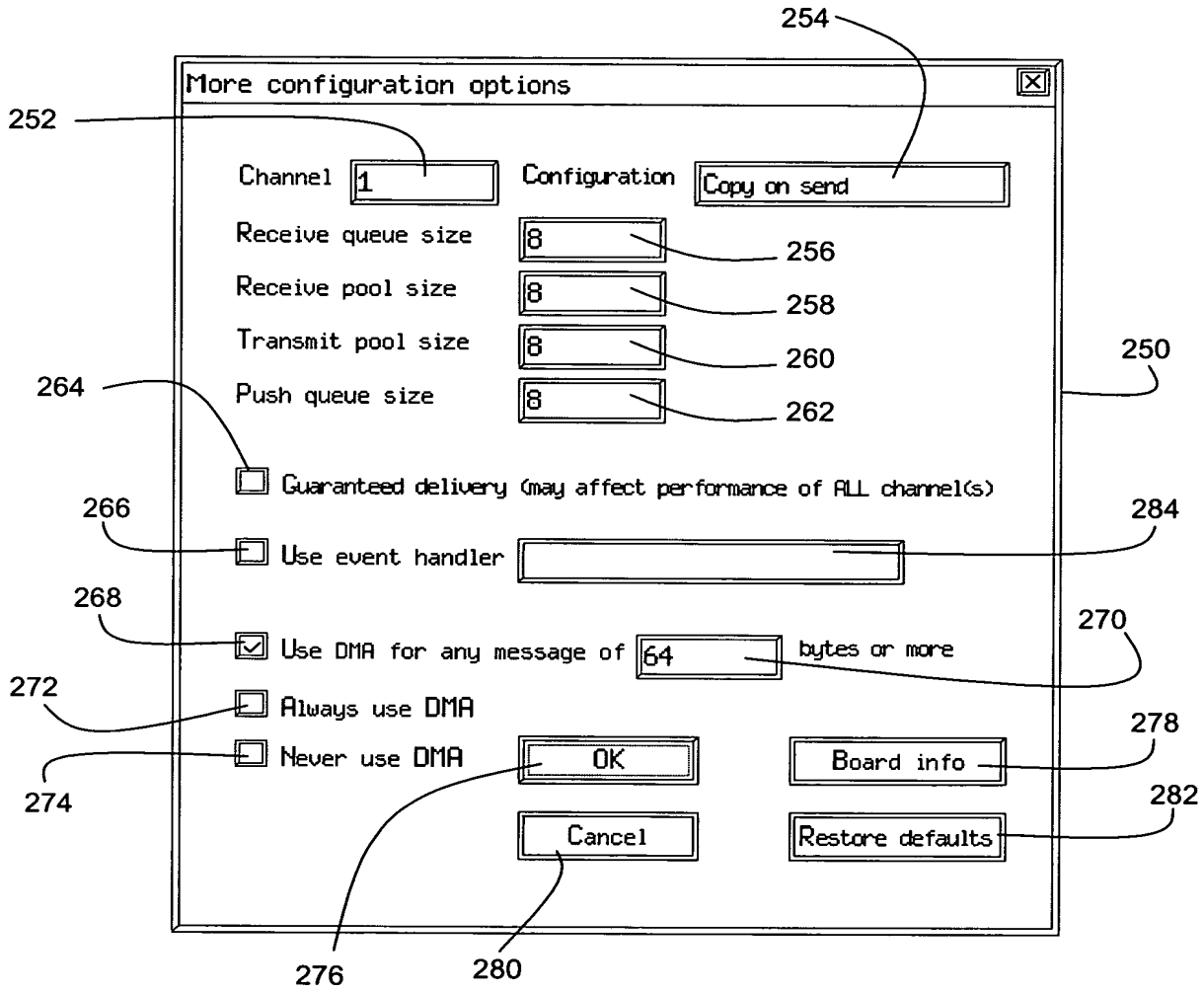

Figure 23: The "More configuration options" dialog window

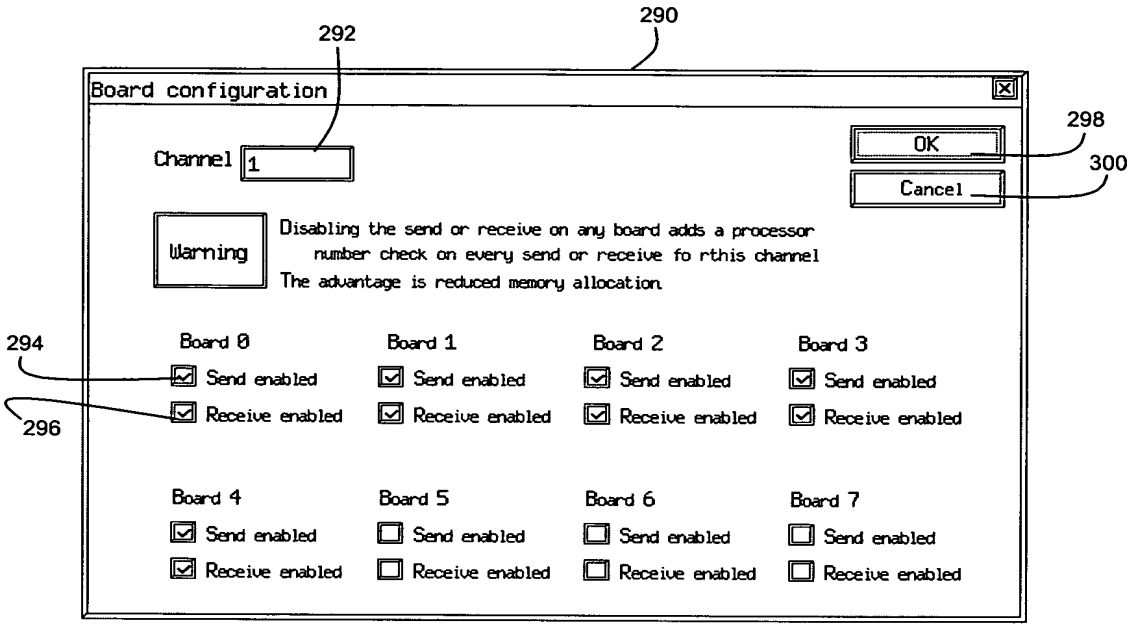

Figure 24: The "Board configuration" dialog window

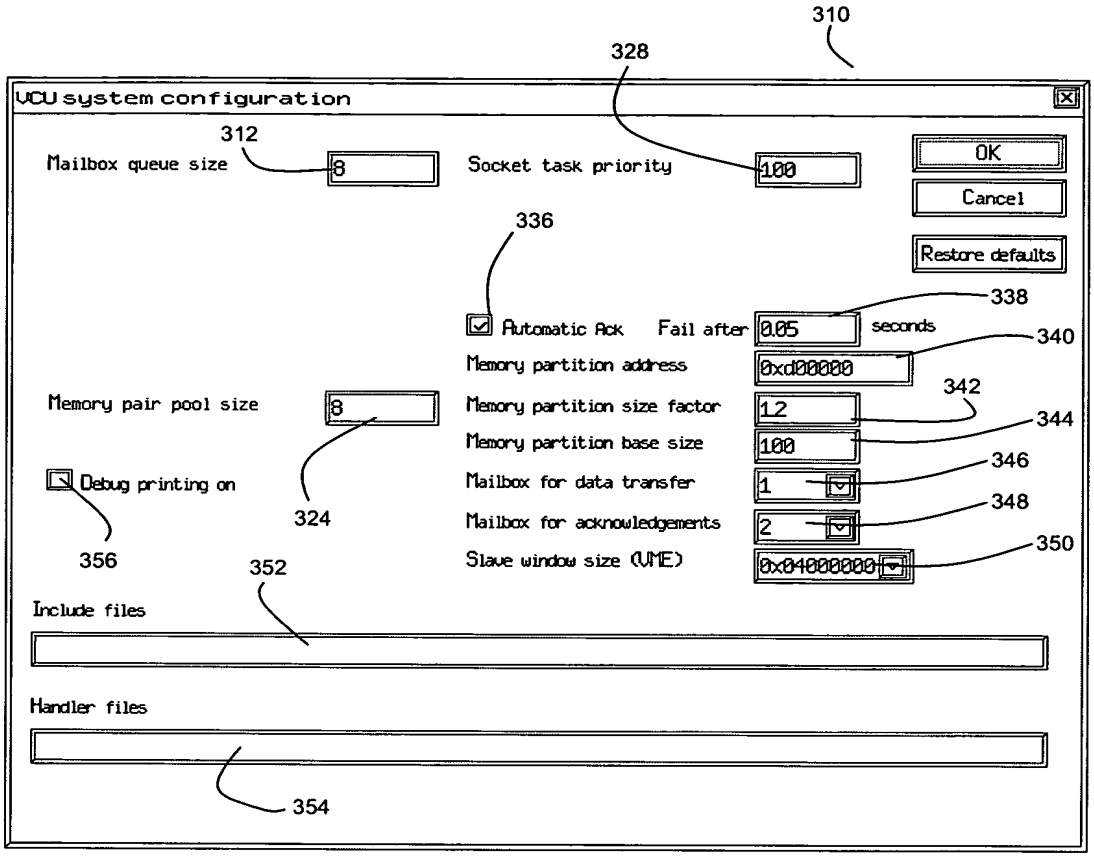

Figure 25: The "VCU system configuration" dialog window

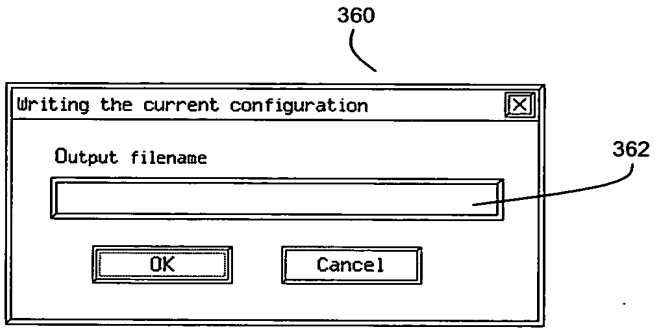

Figure 26: The "Writing the current configuration" dialog window

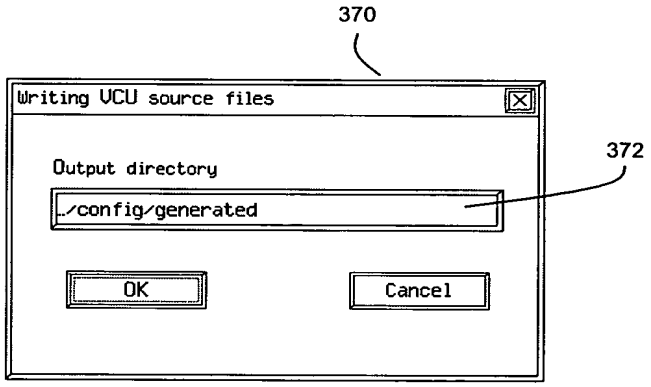

Figure 27: The "Writing the VCU source files" dialog window

Configuration:	Receive Queue Size:	Receive Pool Size:
Copy on Send	Maximum difference between the number of remote vcuSend() and local vcuRecv() calls. On overflow, the message is lost or the entire system blocks	Maximum difference between the number of remote vcuSend() and local vcuRecv() calls. On overflow, the message is lost or the entire system blocks
Copy to Pool on Receive	Maximum difference between the number of remote vcuSend() and local vcuRecv() calls. On overflow, the message is lost or the entire system blocks.	Maximum difference between the number of vcuRecv() and vcuFree() calls. On overflow, vcuRecv() returns an error
Copy to Buffer on Receive	Maximum difference between the number of remote vcuSend() and local vcuRecv() calls. On overflow, the message is lost or the entire system blocks	N/A
Push to Pool on Receive	Maximum number of remote sources of vcuSend() calls (each source has a unique ID and keeps track of the transmit pool address it is using). On overflow, the message is lost or the entire system blocks.	Maximum difference between the number of vcuRecv() and vcuFree() calls. On overflow, vcuRecv() returns an error
Push to Buffer on Receive	Maximum number of remote sources of vcuSend() calls (each source has a unique ID and keeps track of the transmit pool address it is using). On overflow, the message is lost or the entire system blocks.	N/A
Queue on Send	Maximum difference between the number of remote vcuSend() and local vcuRecv() calls. On overflow, the message is lost or the entire system blocks	N/A
Copy to Self	Maximum difference between the number of vcuSend() and vcuRecv() calls. On overflow, vcuSend() returns an error	Maximum difference between the number of vcuRecv() and vcuFree() calls. On overflow, vcuRecv() returns an error
Overwrite on Send	N/A	N/A

Figure 28

Configuration:	Transmit Pool Size:	Push Queue Size:
Copy on Send	Maximum number of local sources of vcuSend() calls. On overflow, vcuSend() returns an error.	0
Copy to Pool on Receive	Maximum difference between the number of local vcuSend() and remote vcuRecv() calls. On overflow, vcuSend() returns an error.	0
Copy to Buffer on Receive	Maximum difference between the number of local vcuSend() and remote vcuRecv() calls. On overflow, vcuSend() returns an error.	0
Push to Pool on Receive	Twice the number of local sources of vcuSend() calls (each source has a unique ID and keeps track of the transmit pool address it is using). On overflow, vcuSend() returns an error.	Maximum number of local sources of vcuSend() calls. On overflow, vcuSend() returns an error
Push to Buffer on Receive	Twice the number of local sources of vcuSend() calls (each source has a unique ID and keeps track of the transmit pool address it is using). On overflow, vcuSend() returns an error.	Maximum number of local sources of vcuSend() calls. On overflow, vcuSend() returns an error
Queue on Send	Maximum number of local sources of vcuSend() calls. On overflow, vcuSend() returns an error.	0
Copy to Self	N/A	0
Overwrite on Send	Maximum number of local sources of vcuSend() calls. On overflow, vcuSend() returns an error.	0

Figure 29

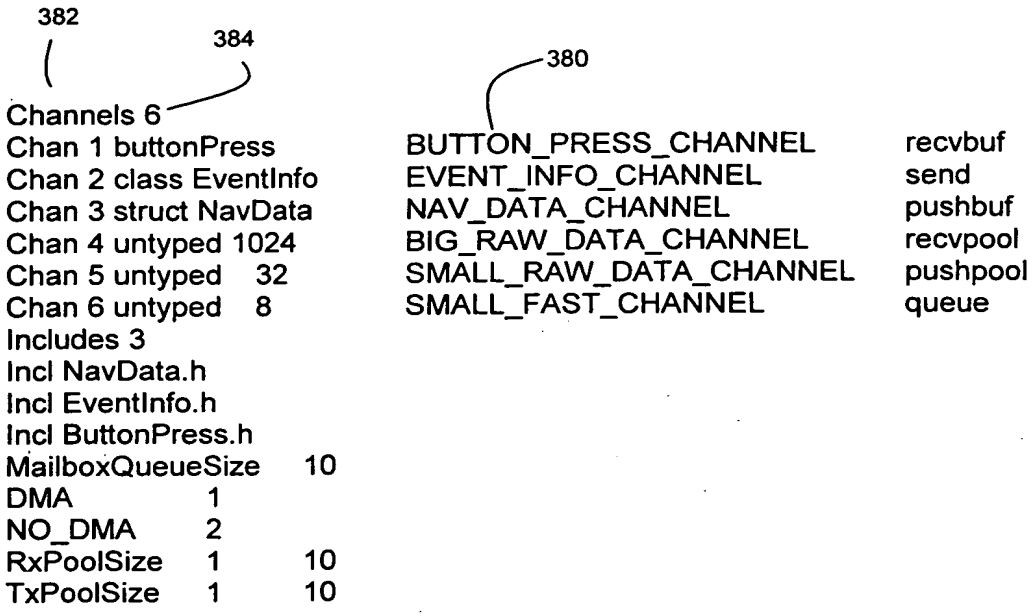

Figure 30: An example configuration file

Keyword:	Parameters:	Meaning:
AckMailbox	1	"AckMailbox" indicates the mailbox used for the automatic acknowledgement. It should differ from the "DataMailbox" value. The default value is 2.
AutoAckOff	0	"AutoAckOff" turns off the automatic acknowledgement system for the VCU. With auto ack turned off, the VCU uses only one mailbox interrupt per board, but cannot guarantee that messages will not be lost.
AutoAckOn	1	"AutoAckOn" turns on the automatic acknowledgement system for the VCU. The parameter sets the delay for an acknowledgement from the receiving board before the message is assumed lost. The default value for the parameter is 0.05 (in seconds).
BoardCount	1	"BoardCount" specifies the number of boards in the system. This value is only used in conjunction with "NoRxPool" and "NoTxPool".
Chan	4 (or 5)	"Chan" specifies a data type, channel name, and configuration type for one channel. Specifying "Chan" a second time for a channel resets the DMA size specs, even if they were changed from default by a "DmaStart" line. "Chan 0" is not allowed. Also, "Chan" cannot be specified before "Channels".
Channels	1	"Channels" specifies the number of channels in the VCU. It should not appear more than once in the configuration file.
DataMailbox	1	"DataMailbox" indicates the mailbox used by the VCU for inter-board communication. It must differ from the "AckMailbox" value. The default value is 1.
DebugOn	0	Turns on printing of all values received in the data mailbox interrupt
DMA	1	"DMA" specifies that the indicated channel should always use DMA.
DmaStart	2	"DmaStart" specifies that the indicated channel should use DMA for any message larger than that specified by the second parameter (the size being in bytes).
Event	2	"Event" specifies a routine that should be called as soon as a board receives data on the indicated channel. The first parameter is the channel, and the second parameter is the routine (the routine preferably cannot take any input argument and cannot return any value).
GuaranteedDelivery	1	"GuaranteedDelivery" indicates that the channel will block on certain queue and pool overflows, rather than lose the message. The queue and pool overflows in question are those that are unreportable as returns to vcuSend(), vcuRecv(), or vcuFree() calls. Blocking on these overflows blocks all incoming messages for the board in question and can result in other queues and pools overflowing.
Handlers	1	"Handlers" specifies the number of files to be included in the vcuConfig.cpp file. It should not be called twice.
Hndlr	1	"Hndlr" specifies a file to be included in the vcuConfig.cpp file. If too many or too few files are specified, the autoConfig code declares the configuration invalid. "Hndlr" should not be specified before "Handlers".
Includes	1	"Includes" specifies the number of files to be included in the file vcuInterface.h. "Includes" should not be called more than once.
Incl	1	"Incl" specifies a file to be included in the file vcuInterface.h. If too many or too few files are specified, the autoConfig code declares the configuration invalid. "Incl" should not be specified before "Includes".
MailboxQueueSize	1	"MailboxQueueSize" specifies the size of the msgQ leaving the mailbox ISR. The default value is 8.
MemPairPoolSize	1	"MemPairPoolSize" specifies the size of the memory pool for memory pairs (used in the "push..." configuration types). This is a board-wide memory pool, used by all channels. The default value is 8.

Figure 31(a)

Keyword:	Parameters:	Meaning:
No_DMA	1	"No_DMA" specifies that the indicated channel should never use DMA.
NoRxPool	2	"NoRxPool" specifies a channel and board where the VCU should not support receive operations. The first parameter is the channel and the second parameter is the board's system processor number. If NoRxPool is specified for any board in a channel, then the system processor numbers should be in the range of 0 to BoardCount-1.
NoTxPool	2	"NoTxPool" specifies a channel and board where the VCU should not support transmit operations. The first parameter is the channel and the second parameter is the board's system processor number. If NoTxPool is specified for any board in a channel, then the system processor numbers should be in the range of 0 to BoardCount-1.
PartitionAddress	1	"PartitionAddress" specifies the location of the memory partition created by the VCU for VME bus accessibility.
PartitionFactor	1	"PartitionFactor" specifies the factor used in setting the size of the partition. Buffers used to receive data in the "push to buffer on receive" configuration type are allocated from here, so these drive the size of the partition factor. The factor also allows for fragmented allocation because of the memory alignment specified in the allocation.
RxPoolSize	2	"RxPoolSize" specifies the size of the receiving memory pool for a channel. The first parameter is the channel, and the second parameter is the pool size. The default value is 8.
RxQueueSize	2	"RxQueueSize" specifies the size of the receiving queue for a channel. The first parameter is the channel, and the second parameter is the queue size. The default value is 8.
RxReplyQueueSize	1	"RxReplyQueueSize" specifies the size of the queue delivering responses from the VCU Rx Task to the vcuRecv() routine.
RxRqstQueueSize	1	"RxRqstQueueSize" specifies the size of the queue delivering vcuRecv() requests to the VCU Rx Task.
RxTaskCount	1	"RxTaskCount" specifies the number of tasks set aside to handle vcuRecv() calls. The first task handles all non-blocking calls, while the other tasks handle blocking calls. The minimum value should be 2 unless vcuRecv() will never be called as blocking. The default value is 4.
SlaveWindowSize	1	"SlaveWindowSize" specifies the size of the slave windows for boards in the system. The valid options are "half" and "full", which correspond to 0x04000000 and 0x08000000. The default value is "half".
TaskPriority	2	"TaskPriority" specifies the priority of the indicated task, "Socket", "Tx", "Rx", or "Debug". The default value is 100 for all tasks.
TxPoolSize	2	"TxPoolSize" specifies the size of the transmitting memory pool for a channel. The first parameter is the channel, and the second parameter is the pool size. The default value is 8.
TxReplyQueueSize	1	"TxReplyQueueSize" specifies the size of the queue that delivers update responses from the VCU Tx Task to the vcuSend() routine.
TxRqstQueueSize	1	"TxRqstQueueSize" specifies the size of the queue that delivers vcuSend() requests to the task that actually performs the data transmission (the VCU Tx Task).
TxStatusQueueSize	1	"TxStatusQueueSize" specifies the size of the queue that delivers status responses from the VCU Tx Task to the vcuSend() routine.

Figure 31(b)

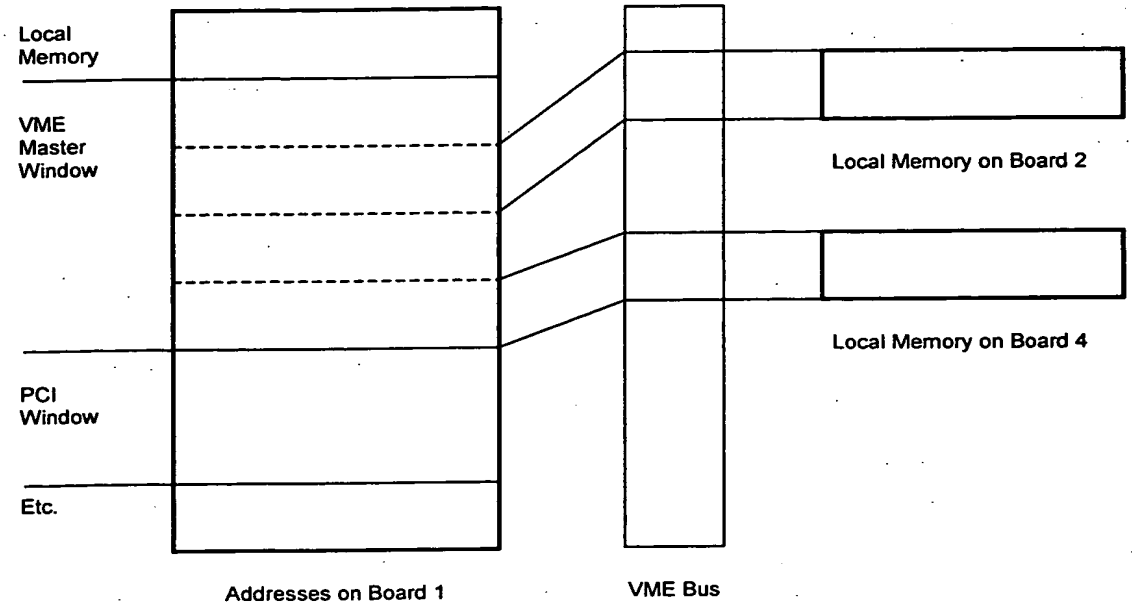
Local Memory
VME Master Window
PCI Window
Etc.
Addresses on Board 1
VME Bus
Local Memory on Board 2
Local Memory on Board 4

Figure 32

Constant:	Value:	Meaning:
VCU_ERROR_NO_CODE	0x1	Mainly used internally, but also used as a return value for a multicast "vcuRequestErrno()" so this routine can be used to check whether any errors exist in the system
VCU_SUCCESS	0x2	The VCU command operated successfully
VCU_RECV_EMPTY	0x3	The non-blocking vcuRecv() command found no message.
VCU_BAD_CHANNEL	0x4	An unexpected channel was received. This can be directly, from an API call, or indirectly, from data corrupted when copied across the VME bus.
VCU_MSGQ_FULL	0x5	Used internally on a vcuSend() when the channel is configured for copy to self.
VCU_POOL_FULL	0x6	The vcuSend() or vcuRecv() call could not find sufficient memory space in the memory pool to perform the send or receive.
VCU_NO_RECV_TASK	0x7	The vcuRecv() call is a blocking call, but no task is available for a blocking receive call. To fix: configure the system to have more receive tasks.
VCU_ERR_ON_RECV	0x10000	The error occurred in a section of code responsible for receiving. The code is most likely specific to handling vcuRecv().
VCU_ERR_ON_SEND	0x20000	The error occurred in code specific to handling vcuSend().
VCU_ERR_ON_FREE	0x30000	The error occurred in code specific to handling vcuFree ().
VCU_ERR_ON_PING	0x40000	The error occurred in code specific to handling vcuPing().
VCU_ERR_ON_TEST_PAT	0x50000	The error occurred in code specific to handling vcuRequestTestPattern().
VCU_ERR_ON_CONFIG	0x60000	The error occurred at initialization
VCU_ERR_ON_ERRNO	0x70000	The error occurred in code specific to handling vcuRequestErrno(). These error numbers are not stored, so that stored errors can be differentiated from errors that occur at the time of the request
VCU_ERR_IN_SOCKET	0x01000	The error occurred in code in VcuSocket.cpp.
VCU_ERR_IN_RXQUEUE	0x02000	The error occurred during an RxQueues method.
VCU_ERR_IN_MAILBOX	0x03000	The error occurred in mailbox management, in the file VcuMailbox.cpp.
VCU_ERR_IN_INTERFACE	0x04000	The error occurred in the interface to the vcuSocket, in VcuComm.cpp.
VCU_ERR_IN_RX_TASK	0x05000	The error occurred in code in VcuComm.cpp in logic specific to Rx Task management.
VCU_ERR_IN_TX_TASK	0x06000	The error occurred in code in VcuComm.cpp in logic specific to Tx Task management.
VCU_ERR_IN_VME_COPY	0x07000	The error occurred in code responsible for copying data across the VME bus.
VCU_ERR_IN_RXPOOL	0x08000	The error occurred in a RxPools method.
VCU_ERR_IN_TXPOOL	0x09000	The error occurred in a TxPools method.
VCU_ERR_IN_PAIRPOOL	0x0a000	The error occurred in a MemPairPool method.
VCU_ERR_IN_ACK	0x0b000	The error occurred in code responsible for handling an automatic acknowledgement.

Figure 33(a)

Constant:	Value:	Meaning:
VCU_ERR_NO_MEMORY	0x00010	The error occurred when requested memory is unavailable.
VCU_ERR_MSGQ_RECV	0x00020	The error occurred while trying to receive from a msgQ.
VCU_ERR_DATA_SYNCH	0x00030	The error occurred when VcuSocket tried to free a memory pool slot already freed or during RequestDataPush() when the incorrect identifier came back. The latter requires rewriting the Push communication protocol.
VCU_ERR_INTERNAL	0x00040	The error occurred when the flow reached a section of code it should not have reached.
VCU_ERR_BAD_POOL_SLOT	0x00050	The error occurred when the VcuSocket tried to Free() memory that did not come from the pool specified.
VCU_ERR_BAD_MAILBOX	0x00060	The error occurred when the VcuSocket tried to TriggerRemoteBoard() to itself or a destination outside the system boundaries.
VCU_ERR_BAD_MCAST	0x00070	The error occurred because the VCU was reconfigured so that the destinations are not specified bitwise, or a multicast was attempted on a channel configured for updating ("push ...")
VCU_ERR_UNLOCK_FAILED	0x00080	The error occurred during Update() when the system was unable to unlock a memory slot it just locked. This should never occur.
VCU_ERR_MSGQ_FULL	0x00090	The error occurred when a send to a msgQ found the msgQ full.
VCU_ERR_MSGQ_SEND	0x000a0	The error occurred while trying to send to a msgQ. The error is not a full msgQ error.
VCU_ERR_BAD_CONFIG	0x000b0	The error occurred when the VCU tables were in conflict with the kernel configuration.
VCU_ERR_TRIG_FAILED	0x000c0	The error should only occur when automatic acknowledgement is enabled. The error means that the automatic ack was not received. vcuPing() can be tried to make sure that the board is accessible. The return value is VCU_SUCCESS for a successful vcuPing().
VCU_ERR_MSG_TOO_BIG	0x000d0	The error occurred when the size of the message was larger than the capacity of the channel.
VCU_ERR_DEST_INVALID	0x000e0	The error occurred when a vcuPing() or vcuRequestTestPattern() found an invalid destination specified.
VCU_ERR_TASK_DIED	0x000f0	The error occurred when a receive task is found to be inaccessible.
VCU_ERR_VME_DMA	0x00100	The error occurred when a sysVmeDmaCopy() returned an error value.
VCU_ERR_ISR_USE	0x00200	The error occurred when vcuSend() was called from an ISR.
VCU_MBOX_FULL_QUEUE	0xff	The error occurred when the receiver's mailbox ISR found its outgoin msgQ full. The message is lost, but the sender gets an error message.
VCU_MBOX_QUEUE_ERROR	0xfe	The error occurred when the receiver's mailbox ISR found its outgoing msgQ broken (not just full). The message is lost, but the sender gets an error message. All further messages are likely to get the same response.

Figure 33(b)

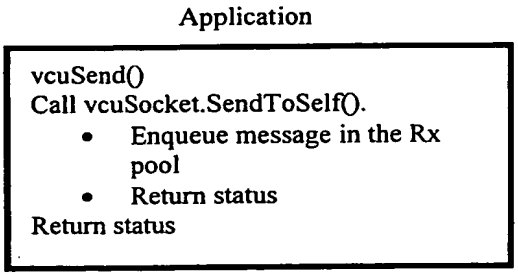

Figure 34: The sending sequence for the "Queue to self" configuration.

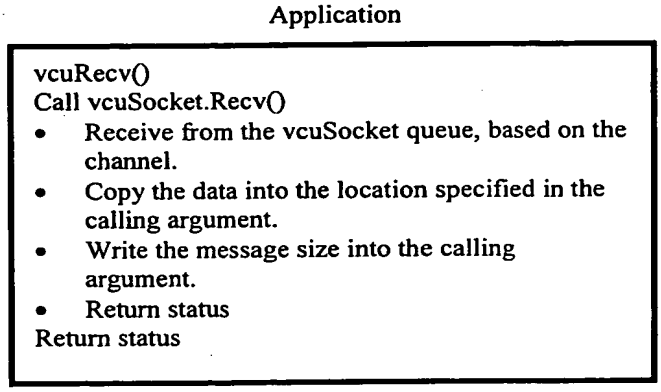

Figure 35: The receiving sequence for the "Queue to self" configuration

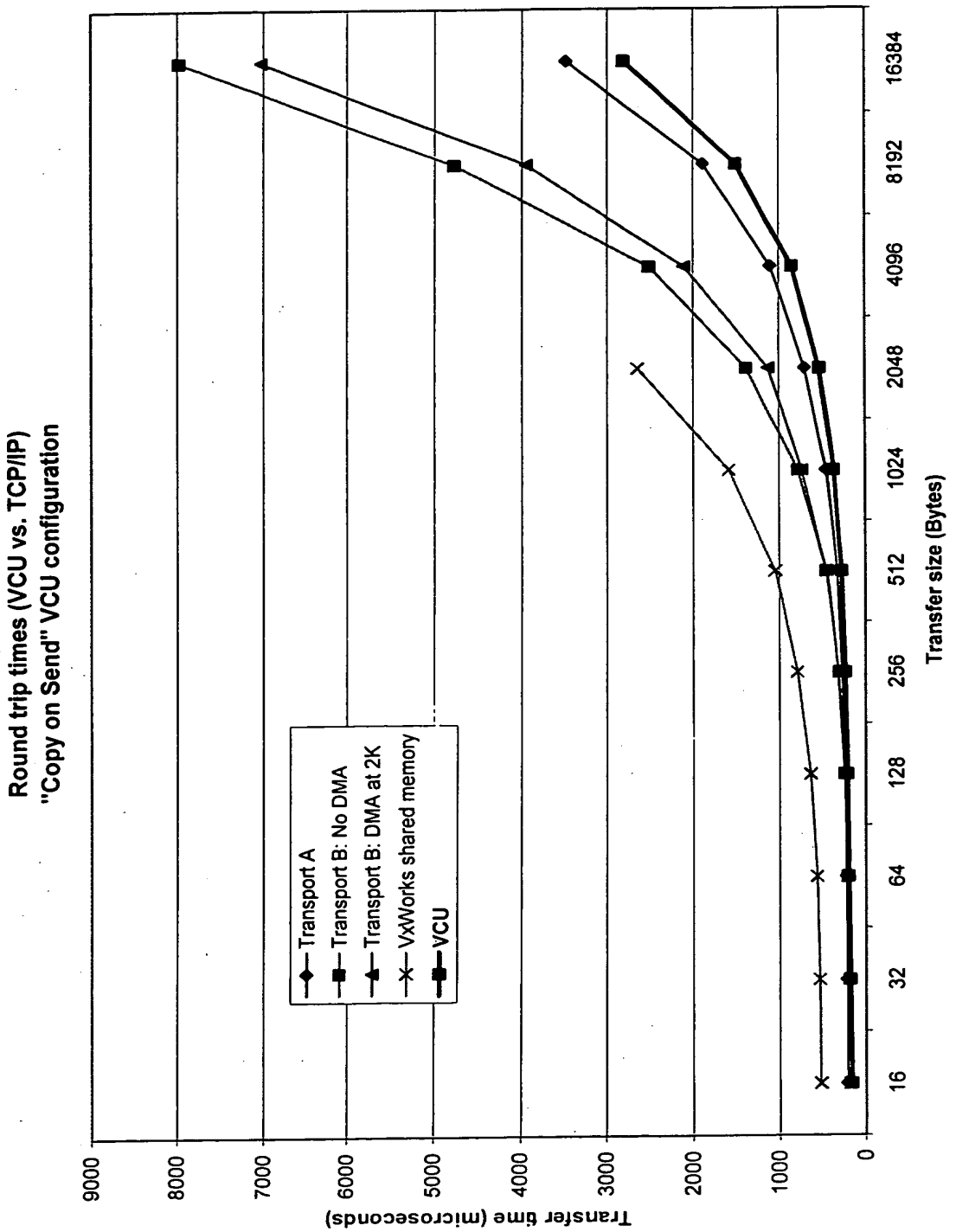
Round trip times (VCU vs. TCP/IP)
"Copy on Send" VCU configuration
Transfer time (microseconds)
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
16
32
64
128
256
512
1024
2048
4096
8192
16384
Transfer size (Bytes)
Transport A
Transport B: No DMA
Transport B: DMA at 2K
VxWorks shared memory
VCU

Figure 36

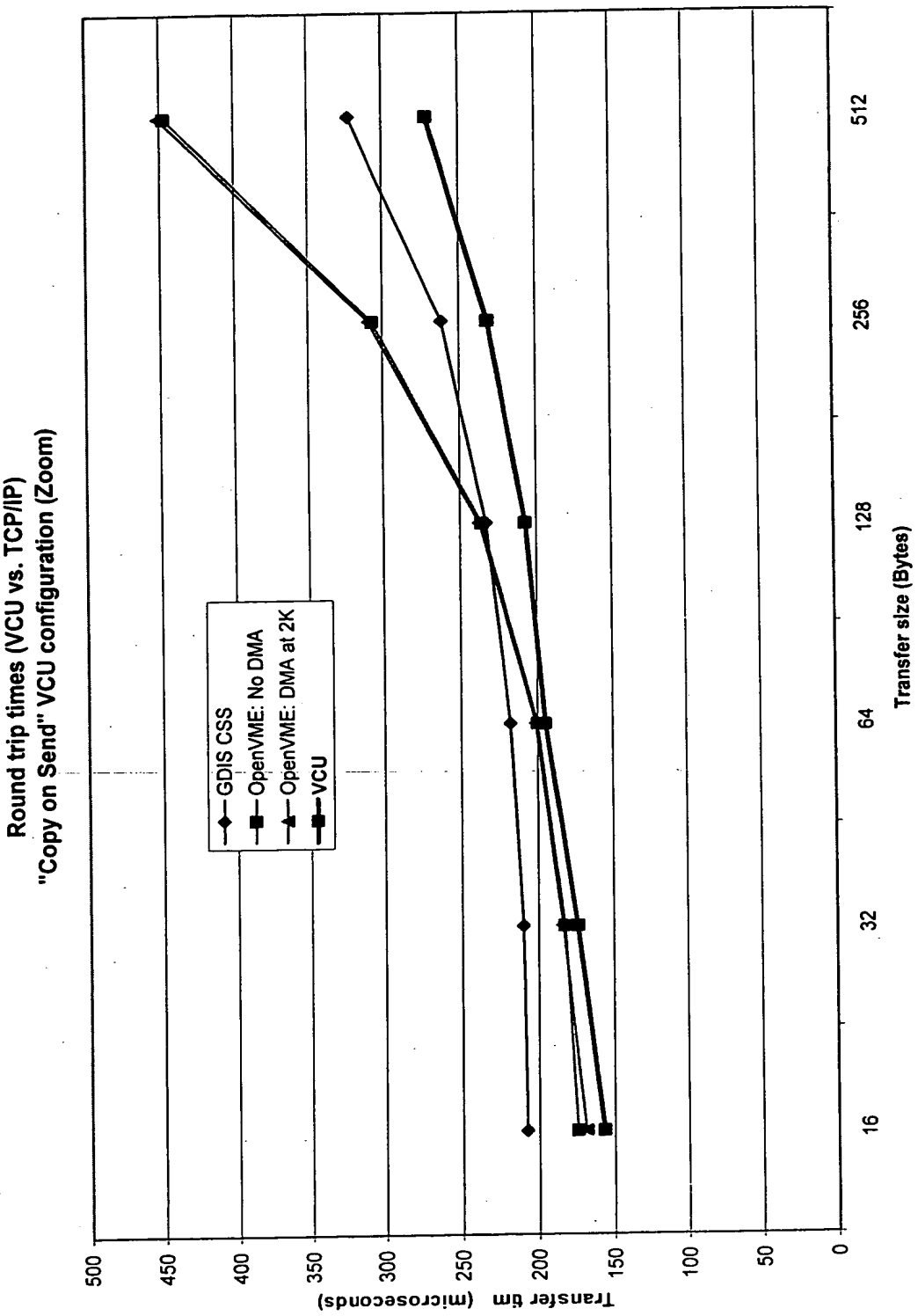
Round trip times (VCU vs. TCP/IP)
"Copy on Send" VCU configuration (Zoom)
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VCU
Transfer tim (microseconds)
500
450
400
350
300
250
200
150
100
50
0
16
32
64
128
256
512
Transfer size (Bytes)

Figure 37

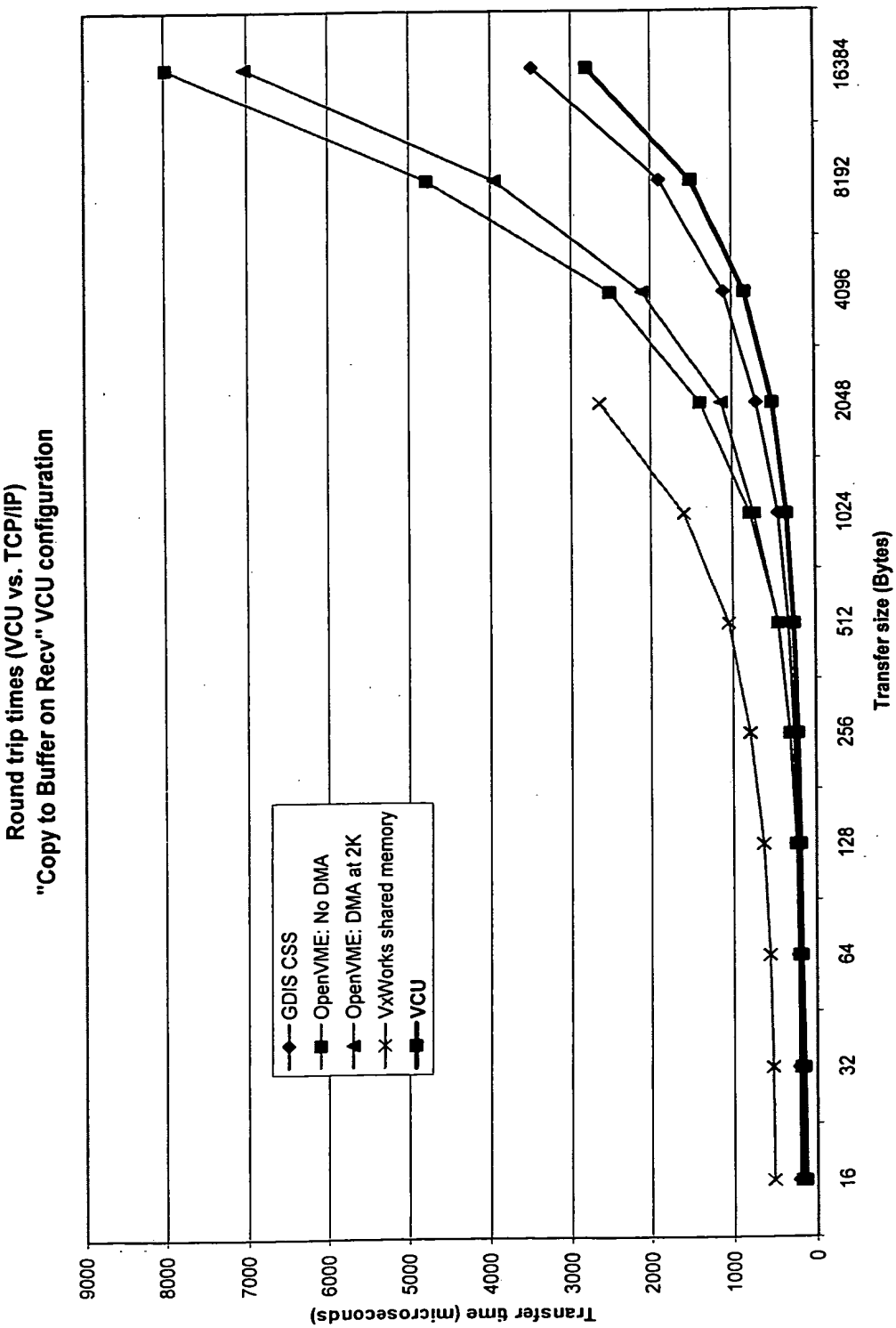
Round trip times (VCU vs. TCP/IP)
"Copy to Buffer on Recv" VCU configuration
Transfer time (microseconds)
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Transfer size (Bytes)
16
32
64
128
256
512
1024
2048
4096
8192
16384
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VxWorks shared memory
VCU

Figure 38

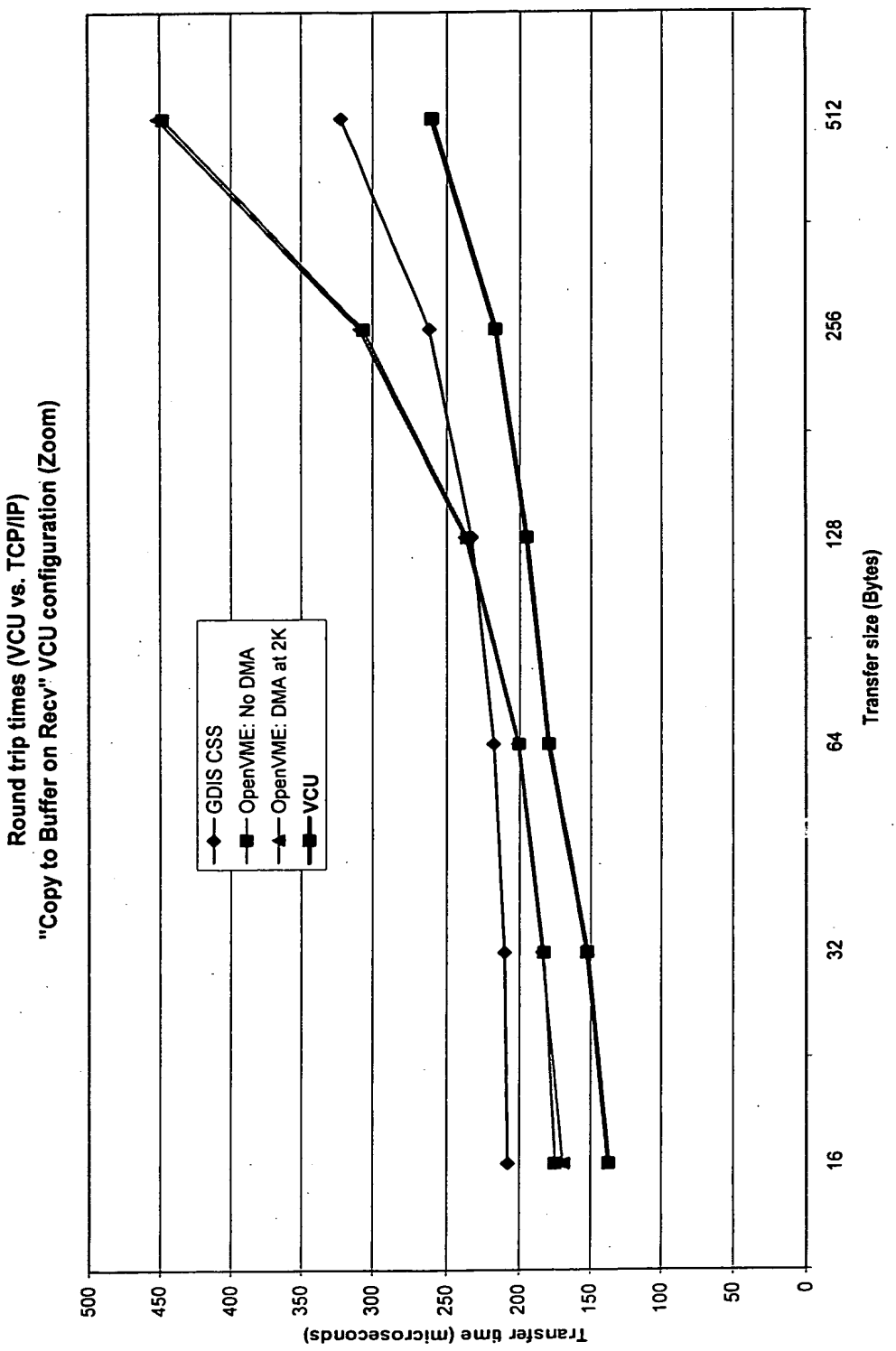
Round trip times (VCU vs. TCP/IP)
"Copy to Buffer on Recv" VCU configuration (Zoom)
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VCU
Transfer time (microseconds)
500
450
400
350
300
250
200
150
100
50
0
16
32
64
128
256
512
Transfer size (Bytes)

Figure 39

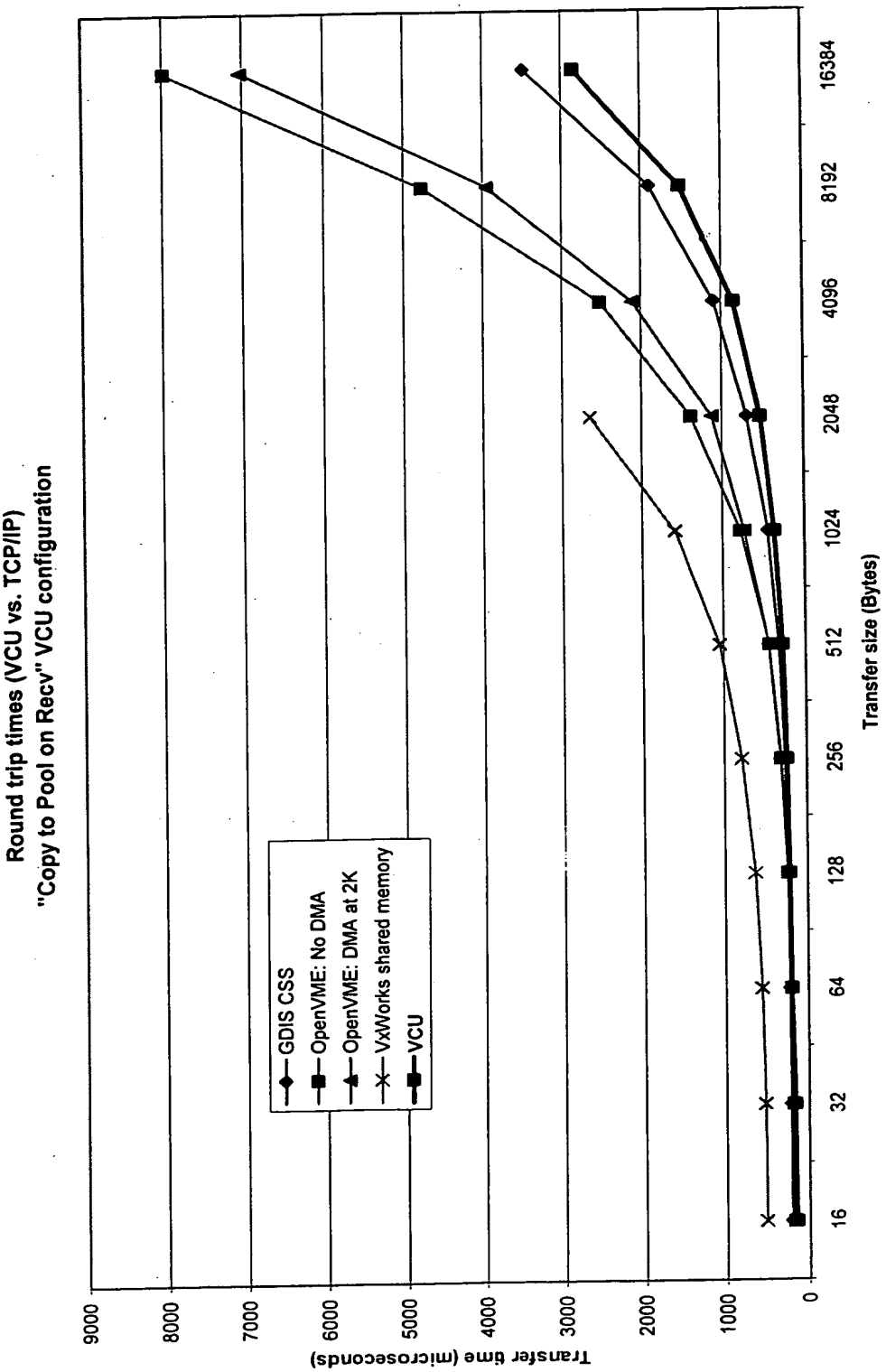
Round trip times (VCU vs. TCP/IP)
"Copy to Pool on Recv" VCU configuration
Transfer time (microseconds)
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Transfer size (Bytes)
16
32
64
128
256
512
1024
2048
4096
8192
16384
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VxWorks shared memory
VCU

Figure 40

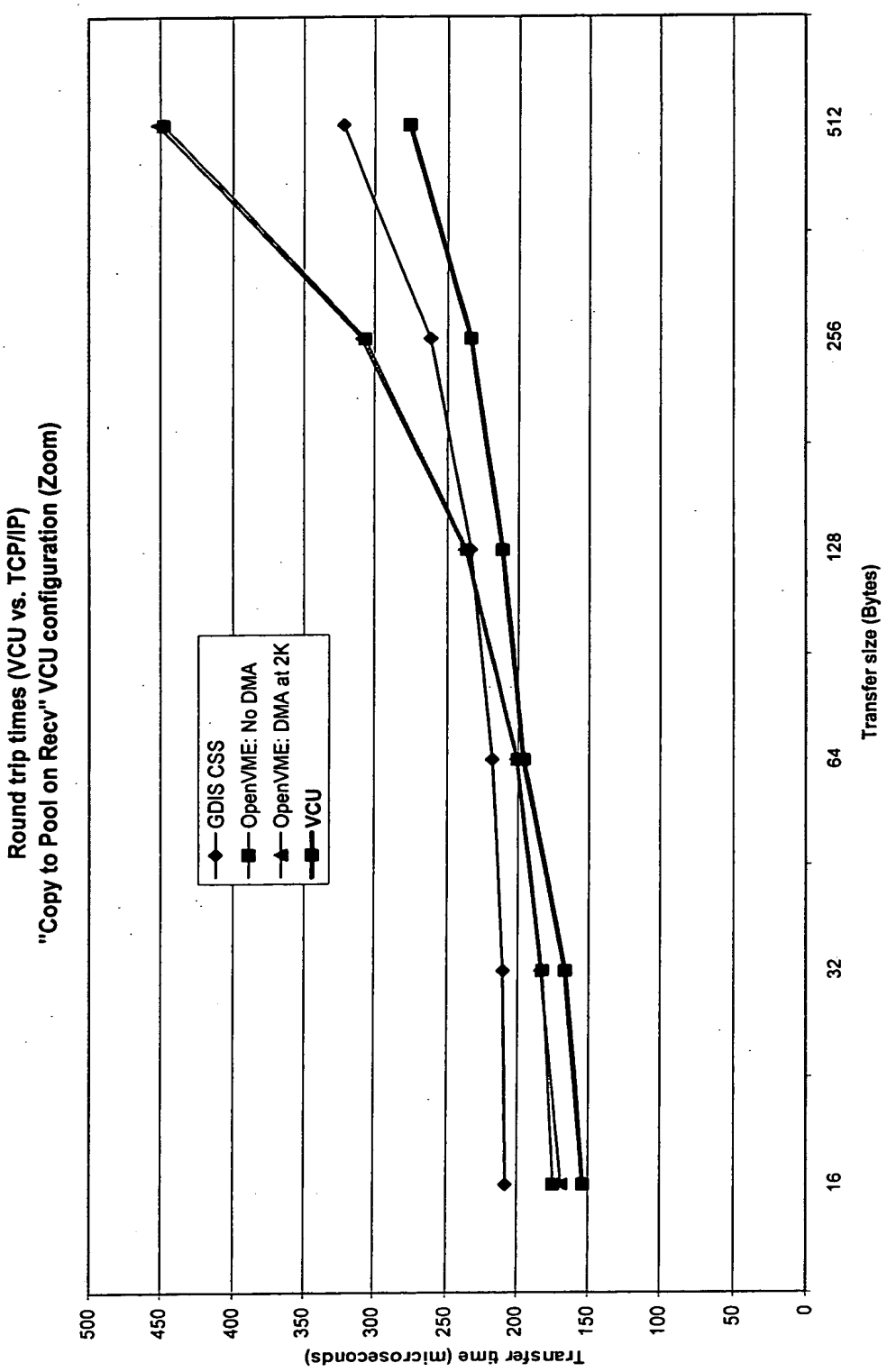
Round trip times (VCU vs. TCP/IP)
"Copy to Pool on Recv" VCU configuration (Zoom)
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VCU
Transfer time (microseconds)
500
450
400
350
300
250
200
150
100
50
0
16
32
64
128
256
512
Transfer size (Bytes)

Figure 41

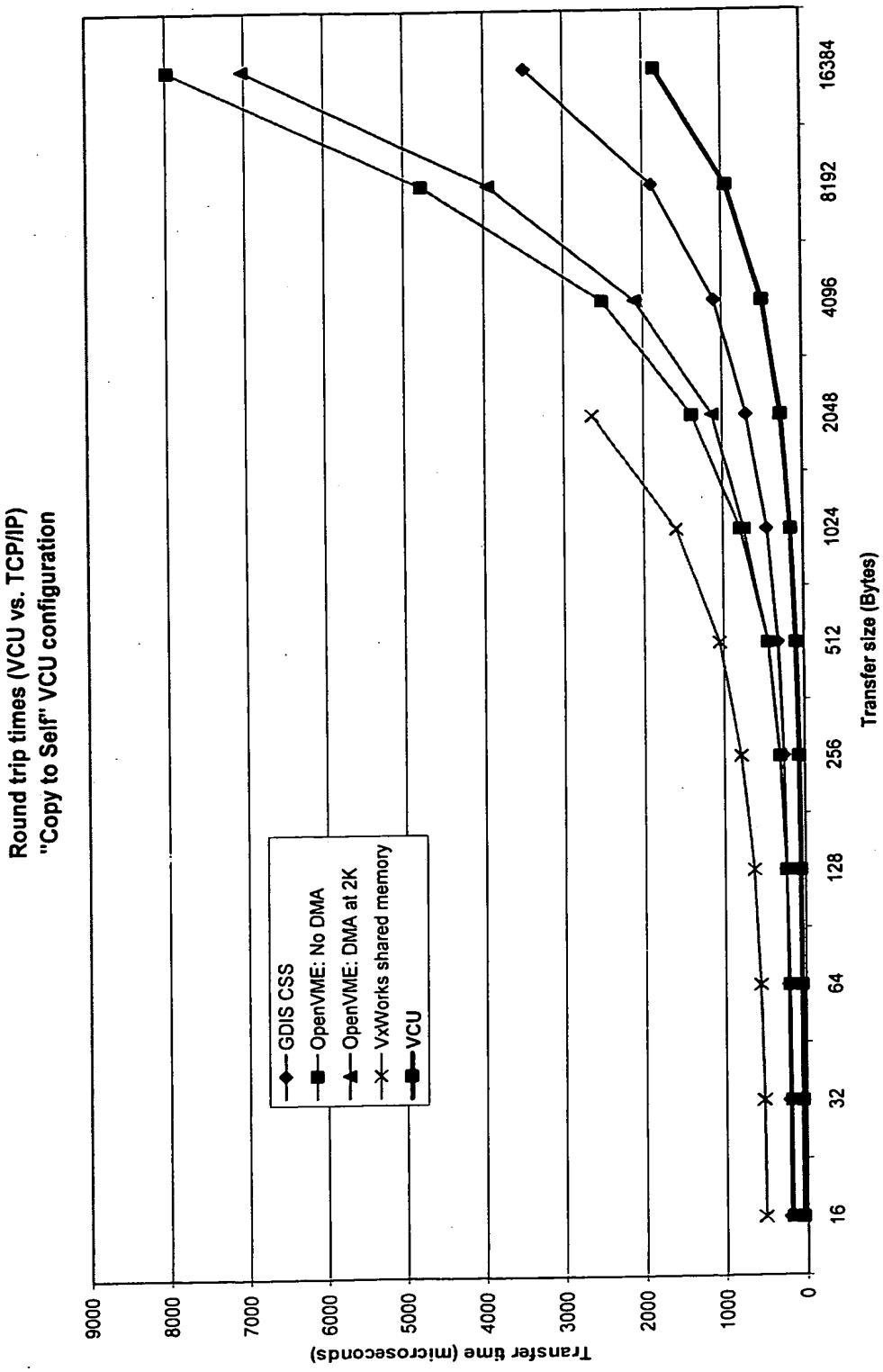
Round trip times (VCU vs. TCP/IP)
"Copy to Self" VCU configuration
Transfer time (microseconds)
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Transfer size (Bytes)
16
32
64
128
256
512
1024
2048
4096
8192
16384
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VxWorks shared memory
VCU

Figure 42

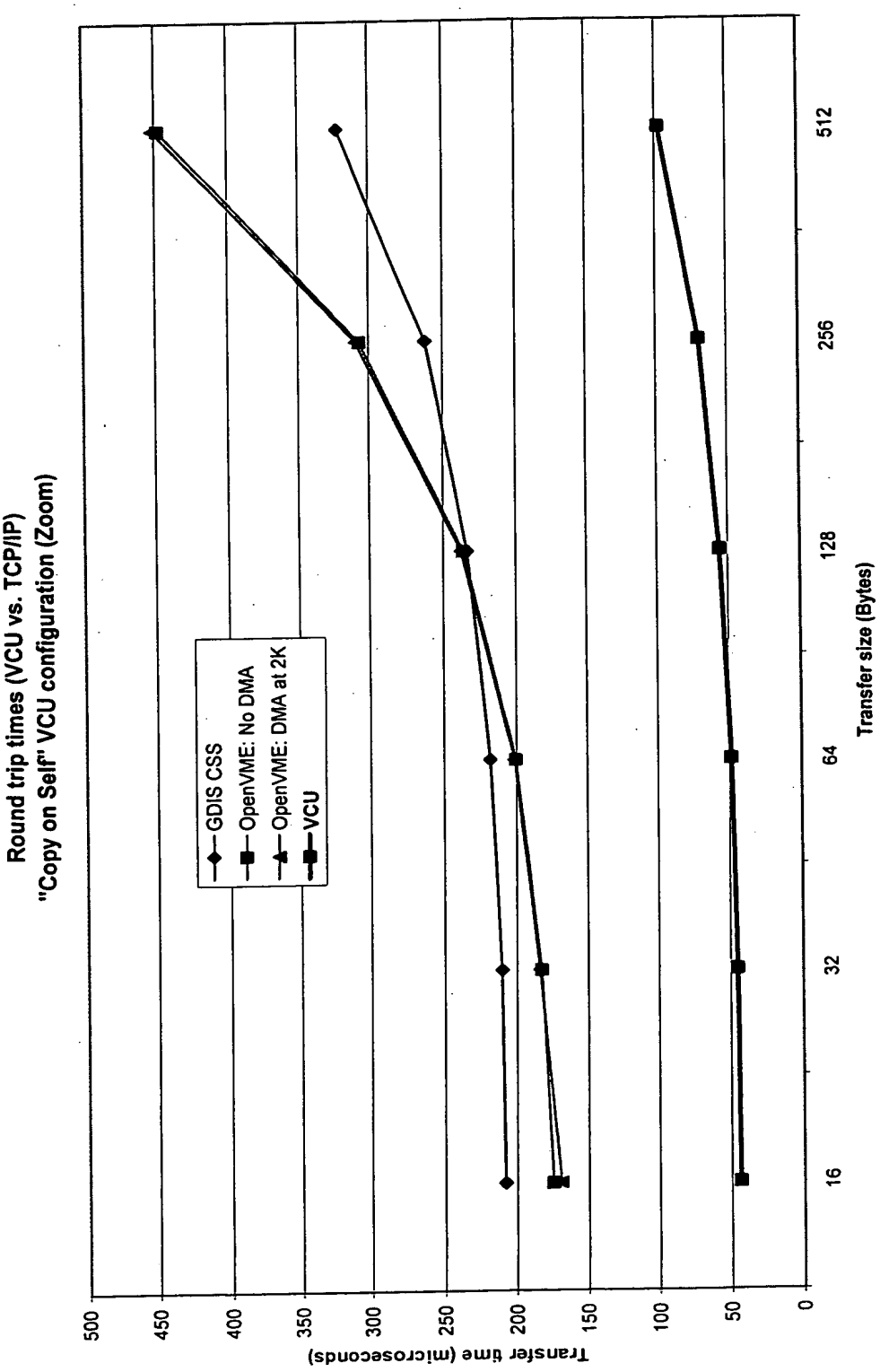
Round trip times (VCU vs. TCP/IP)
"Copy on Self" VCU configuration (Zoom)
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VCU
Transfer time (microseconds)
500
450
400
350
300
250
200
150
100
50
0
Transfer size (Bytes)
16
32
64
128
256
512

Figure 43

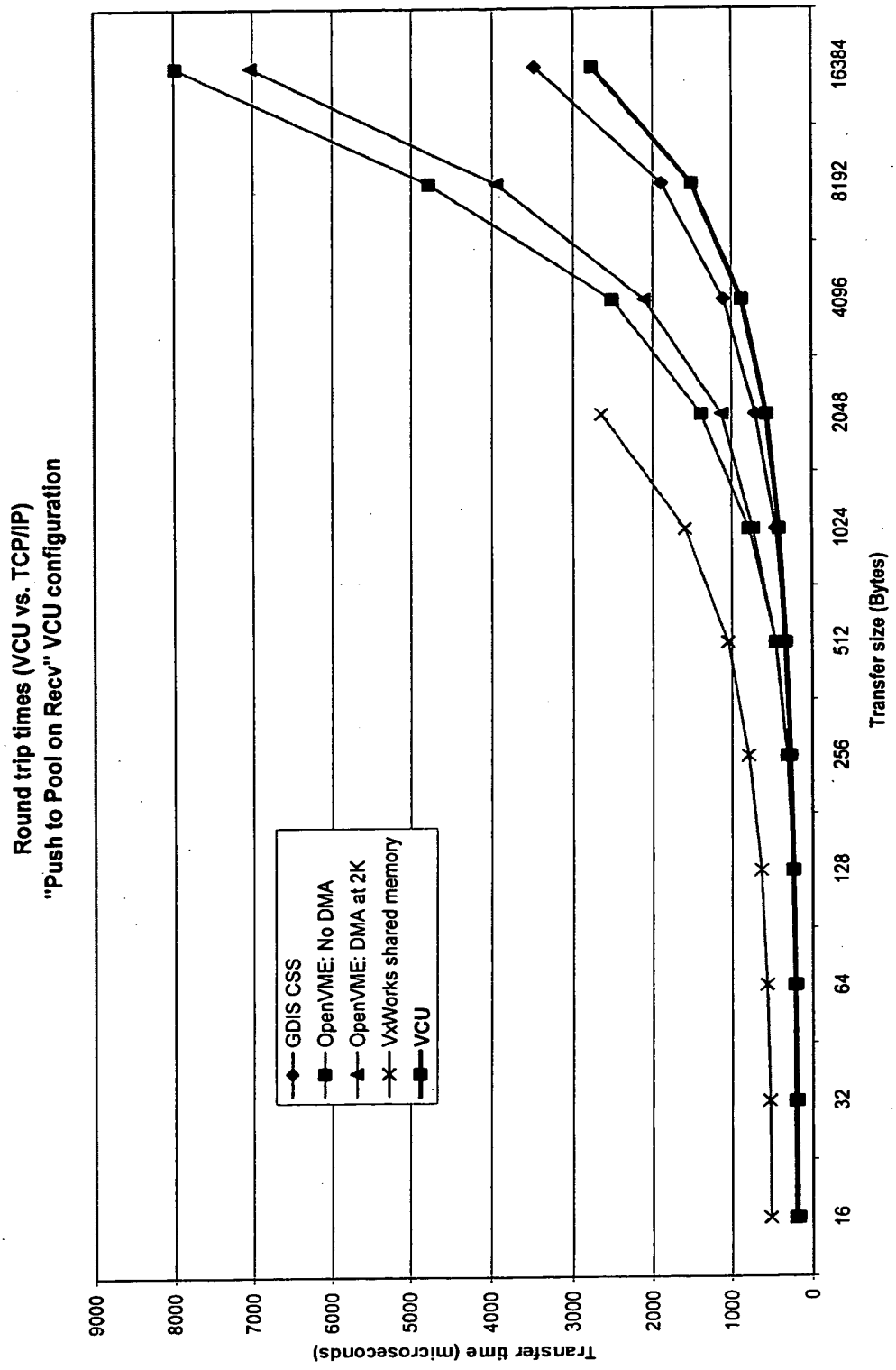
Round trip times (VCU vs. TCP/IP)
"Push to Pool on Recv" VCU configuration
Transfer time (microseconds)
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
16
32
64
128
256
512
1024
2048
4096
8192
16384
Transfer size (Bytes)
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VxWorks shared memory
VCU

Figure 44

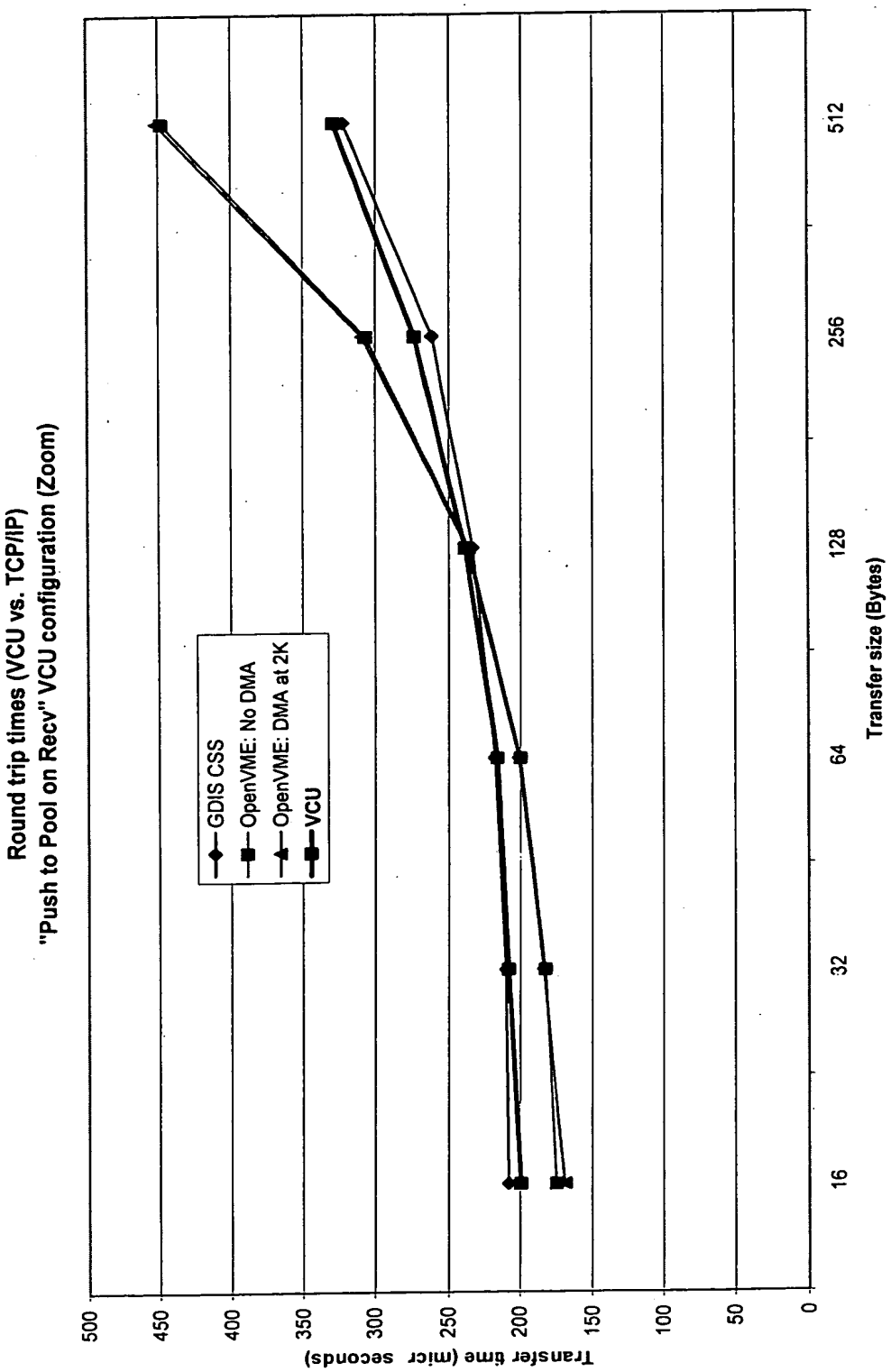
Round trip times (VCU vs. TCP/IP)
"Push to Pool on Recv" VCU configuration (Zoom)
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VCU
Transfer time (micr seconds)
500
450
400
350
300
250
200
150
100
50
0
16
32
64
128
256
512
Transfer size (Bytes)

Figure 45

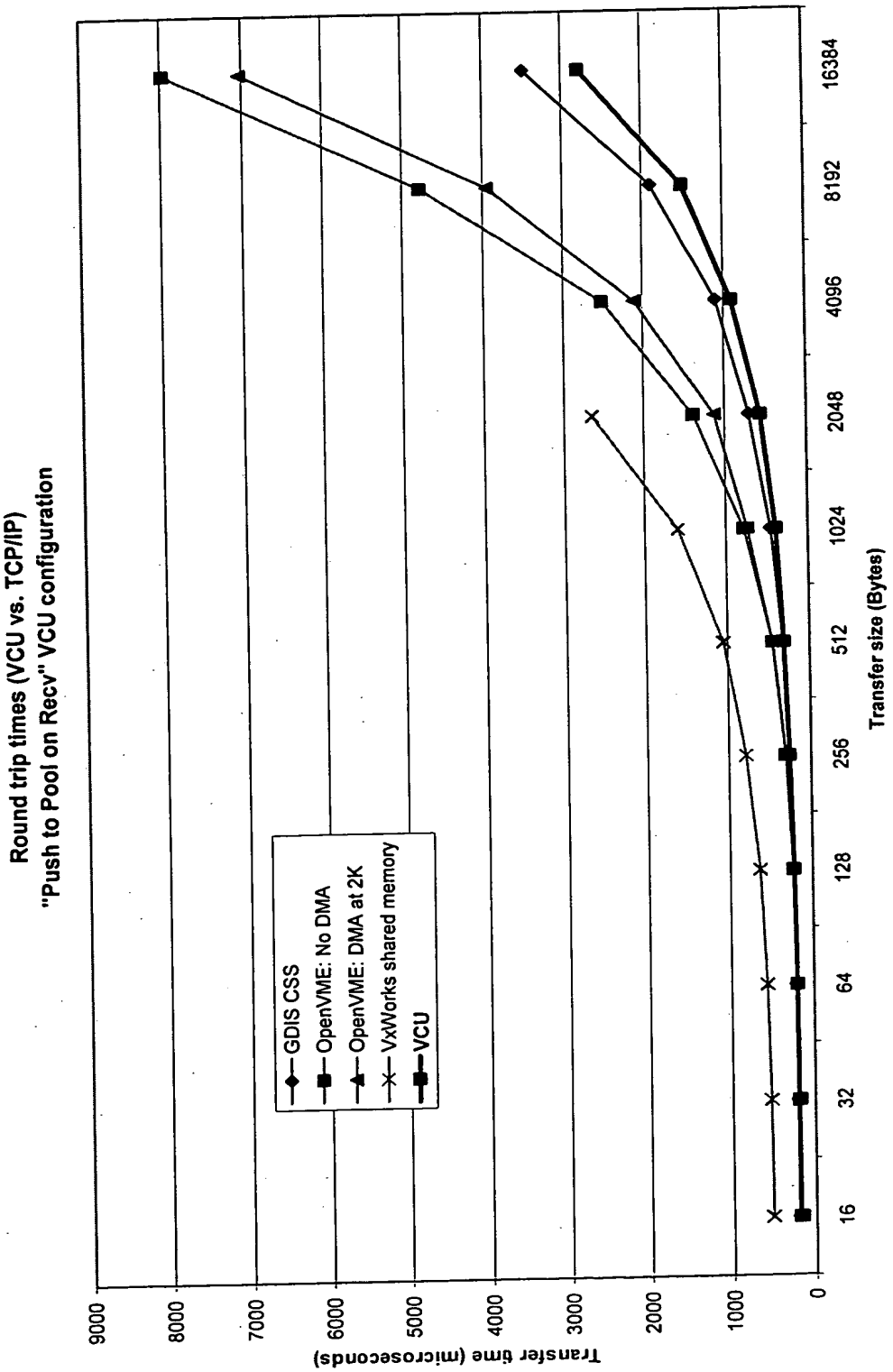
Round trip times (VCU vs. TCP/IP)
"Push to Pool on Recv" VCU configuration
Transfer time (microseconds)
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Transfer size (Bytes)
16
32
64
128
256
512
1024
2048
4096
8192
16384
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VxWorks shared memory
VCU

Figure 46

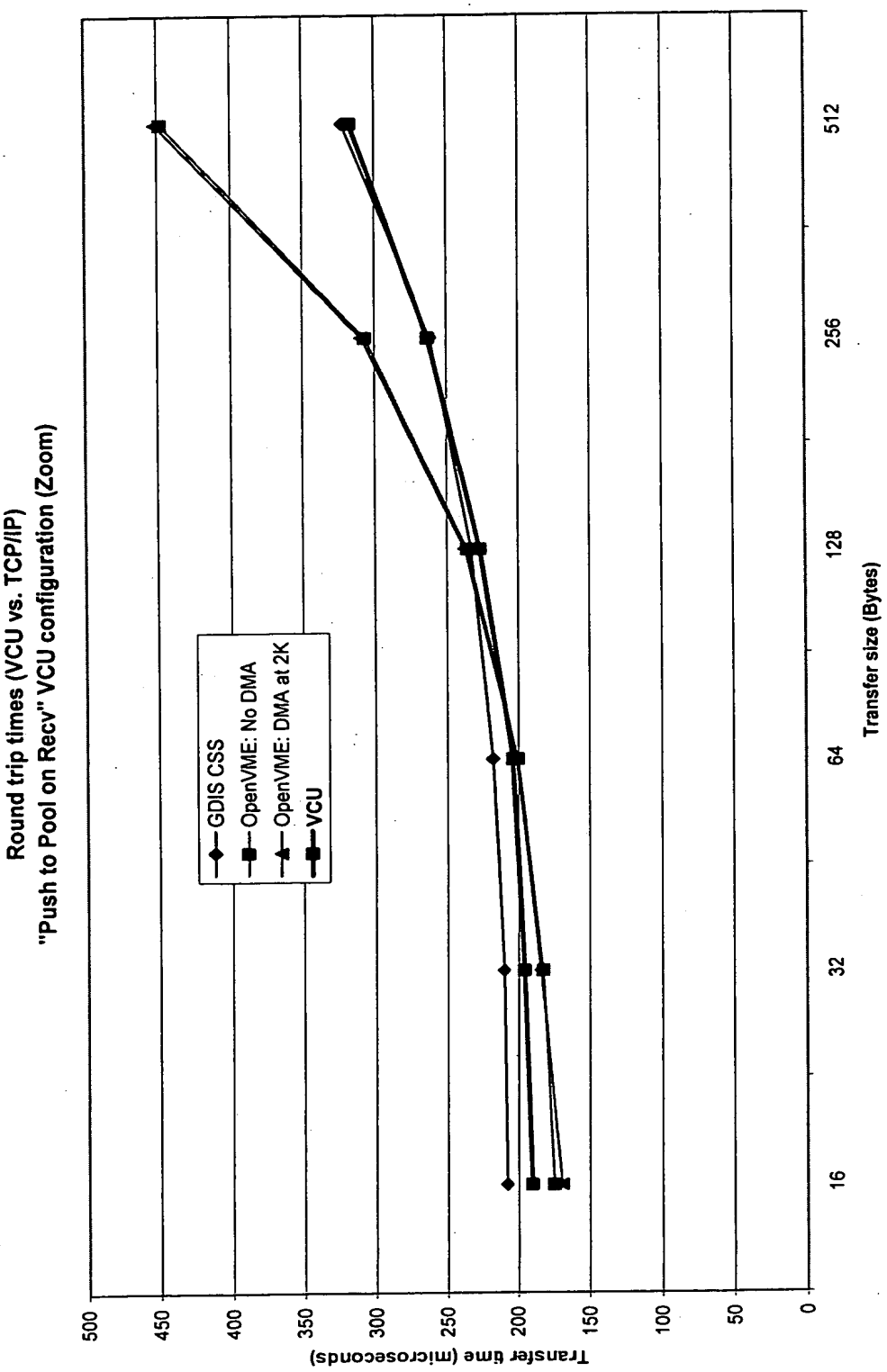
Round trip times (VCU vs. TCP/IP)
"Push to Pool on Recv" VCU configuration (Zoom)
GDIS CSS
OpenVME: No DMA
OpenVME: DMA at 2K
VCU
Transfer time (microseconds)
500
450
400
350
300
250
200
150
100
50
0
16
32
64
128
256
512
Transfer size (Bytes)

Figure 47